NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

SUITABILITY OF THE SRC-6E RECONFIGURABLE COMPUTING SYSTEM FOR GENERATING FALSE RADAR IMAGES

by

Kendrick R. Macklin

June 2004

Thesis Advisor:	Neil Rowe
Second Reader:	Douglas Fouts

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2004	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: Suitability of the SRC-6E Reconfigurable Computing System for Generating False Radar Images	**5. FUNDING NUMBERS**
6. AUTHOR(S) Kendrick R. Macklin	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

This thesis evaluates the usefulness of the SRC-6E reconfigurable computing system for a radar signal processing application and documents the process of creating and importing VHDL code to configure the user definable logic on the SRC-6E. The research builds on previous work which implemented a false radar imaging algorithm on the SRC-6E. Data from alternative computational approaches to the same problem are compared to determine the effectiveness of SRC-6E solution. The results show that the SRC-6E provides and effective solution for implementations with greater than 64 range bins. An evaluation of the SRC-6E difficulty of use is conducted, including a discussion of required skills, experience and development times. The algorithm test code is included in the appendices.

14. SUBJECT TERMS Benchmark, Reconfigurable Computing, VHDL, SRC-6E, FPGA, False Radar Target Synthesis	**15. NUMBER OF PAGES** 149
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

SUITABILITY OF THE SRC-6E RECONFIGURABLE COMPUTING SYSTEM FOR GENERATING FALSE RADAR IMAGES

Kendrick R. Macklin
Lieutenant, United States Navy
B.S., San Diego State University, 1997
M.S., Naval Postgraduate School, 2003

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2004

Author:	Kendrick R. Macklin
Approved by:	Neil Rowe Thesis Advisor
	Douglas Fouts Second Reader
	Peter Denning Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis evaluates the usefulness of the SRC-6E reconfigurable computing system, a particular kind of specialized computer, for the radar-signal processing application of generating false radar images. It documents the process of creating and importing VHDL code to configure the user definable logic on the SRC-6E, building on previous work for the SRC-6E. Data from alternative computational approaches to the same problem are compared to determine the effectiveness of a SRC-6E solution. The results show that the SRC-6E provides no advantage until the task is made significantly complex; for this application, this was at greater than 64 range bins. This supports the hypothesis that the algorithm requires too much initialization effort to take much advantage of the parallel processing and pipelining of the SRC-6E. An evaluation of the SRC-6E difficulty of use is conducted, including a discussion of required skills, experience and development times.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGEMENTS

I would like to thank my Advisor Professor Neil Rowe for accepting both me and my topic for thesis research just a few days before I had to transfer. His willingness and flexibility, while knowing that I would be doing the majority of the work remotely and without his direct guidance, demonstrate his devotion to all students. My life would have been much simpler if I had thought to go see him much earlier.

I would also like to acknowledge David Caliga of SRC Computers, Inc. for providing personal assistance in helping me to understand and use the SRC-6E computer.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION AND SUMMARY OF RESEARCH

A. PURPOSE

The specific purpose of this research was to evaluate the performance, correctness, and ease of use of the SRC-6E reconfigurable computing system built by SRC Computers, Inc., for implementing false radar images. The more general purpose was to aid in establishing a broad base of knowledge on what types of applications are appropriate for implementation on this type of machine. The task of generating false radar images provides a readily available yet suitably complex algorithm for implementation on the SRC-6E. The research builds on previous work [1]. The algorithm chosen was based on a custom chip design previously developed by a faculty/student research team at the Naval Postgraduate School which creates false target radar images. A C language program, written by Professor Douglas Fouts, was also available to use as a standard for comparing the accuracy of results throughout the research.

B. SUMMARY OF RESEARCH

Reconfigurable computing is defined as "the capability of reprogramming hardware to execute logic that is designed and optimized for a specific user's algorithms" [2]. The SRC-6E reconfigurable computer is a Linux-based system consisting of two sides labeled A and B which each contain motherboards holding dual Intel P3 Xeon 1-GHz processors, 1.5 gigabytes of memory, and a SNAP interface card. The SNAP card is a custom interface card which plugs into a motherboard DIMM memory slot and provides connections to the MAP board which is located in a third section of the system. A single MAP board consists of two independent

MAPs. MAP, a registered trademark of SRC Computers, Inc., is the name for the custom hardware. Each MAP consists of three Xilinx Virtex-II-series XC2V6000 FPGAs and 24 megabytes of memory. One of the FPGAs is reserved for control logic while the other two, available for user programs, are called "user logic". The memory is split into six equal banks, labeled A through F, of 4 megabytes each. The user FPGAs are connected to a fixed 100-MHz clock.

Code written in the hardware description languages Verilog and/or VHDL can be ported to the SRC-6E with only minor changes. Several support files are required to help the code exploit the user logic. These files primarily describe the interfaces to the code. The algorithm selected for the research described here was written in VHDL and converted for use on the SRC-6E.

To evaluate the effectiveness of the SRC-6E, timing data was collected from several implementations of the false radar target algorithm. The first implementation was the executable created on the SRC-6E which uses the reconfigurable user logic. The second implementation was a C program which performed the same functionality as the VHDL code. This code was compiled and executed on a 3-GHz Pentium 4 system, using 2 gigabytes of DIMM memory and the Windows XP Professional operating system. The machine used was the same one as the previous implementation to allow comparison of results. The third implementation was the same C program running on the 1-GHz Xeon processor on a Linux based SRC-6E, but not using the MAP. Several input data sets were used in testing. Each data value consisted of a 5-bit number, written as two hexadecimal digits, which represented an intercepted radar signal. Data sets containing 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16284, 32786, 65536, 131072, 262144, and 500000 data values were used. Five timing runs were conducted for each data set on all three data sources.

The timing data shows the SRC-6E MAP instruction execution time is fast, even for very large data set sizes. However, the total execution time for the algorithm studied was disappointing. The extra time appeared to represent delays in the system to prepare and transfer the data in and out of the MAP. Nonetheless, comparison of the MAP execution times for the previous 4-bin macro with the new 4-bin macro shows an improvement for large input set sizes, suggesting that a new interface design can be more efficient. Also, as input set size is increased, our timing results begin to converge.

The SRC-6E has a relatively steep learning curve. There are a few examples in the documentation and a very little work using the system. The errors generated by the system during development are not intuitive and cannot be solved without previous experience with solving the same errors. There are no development tools in place to assist novice users in programming the system. So this is another factor to consider in selecting it for use. But for high performance on large data sets, it still appears desirable.

C. REMAINING CHAPTER OUTLINE

The following outlines the remaining chapters which roughly follow the major steps that were taken throughout the research:

- Chapter II provides background information and review of previous research.
- Chapter III discusses the SRC-6E architecture, programming environment, and documentation.

- Chapter IV discusses modifications to the original VHDL macro programmed using VHDL.
- Chapter V discusses porting the VHDL code to SRC-6E environment.
- Chapter VI presents the data collection methods and analysis.
- Chapter VII provides conclusions and future work recommendations.
- Appendix A contains the VHDL code for all range bin models.
- Appendix B contains the support files required to implement the VHDL code on the SRC-6E.

II. BACKGROUND AND PREVIOUS WORK

A. INTRODUCTION

This chapter discusses background information and reviews previous work, including the basics of the false radar imaging algorithm, the benefits and costs of pipelining the algorithm, use of the chip design and C program in both the current and previous research and gives an overview of the major steps taken to improve, expand and test the algorithm using the SRC-6E.

B. FALSE TARGET RADAR IMAGING ALGORITHM

The algorithm works by splitting a false target image into several range bins, as shown in Figure 1 where a ship is split into four range bins.

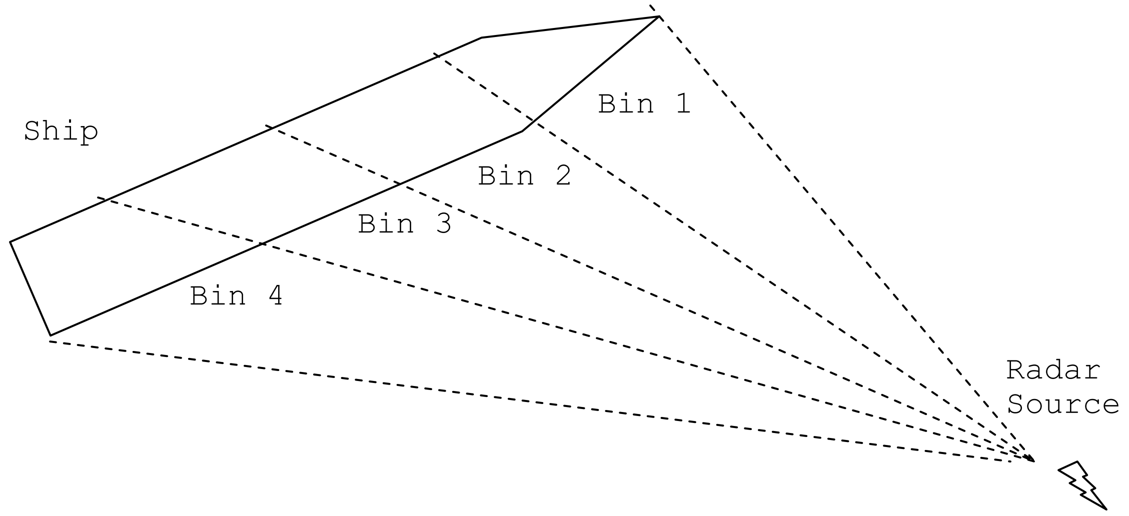

Figure 1. False Target Radar Imaging Algorithm Usage

Each range bin represents a portion of the vessel. Greater resolution can be achieved by having a greater number of range bins for a given false target. Usually the radar-signal travel distance is different for each range bin.

Based on knowledge of a ship’s radar image, an operator can set phase rotation and gain constants for each range bin. The algorithm begins with the interception and sampling of an interrogating radar pulse. The sample phase is then rotated by adding a rotation constant to it. Next, the sine and cosine are calculated. The gain is then applied to the results by multiplying by a gain value. The results of each range bin are then summed up to produce a radar reflection signal at a given time. With proper use, the ship can be made to appear in a false position, be of a different type of target, or to appear to be traveling with other ships.

C. FALSE-TARGET RADAR-IMAGING CHIP DESIGN

We used a custom chip design previously developed by a faculty/student research team at the Naval Postgraduate School which creates false target radar images. The false-target radar-imaging chip consists of a 6-stage pipeline to create a false radar reflection for a single range bin. Figure 2 shows the signal flow through the slightly simplified version that was implemented during the previous research.

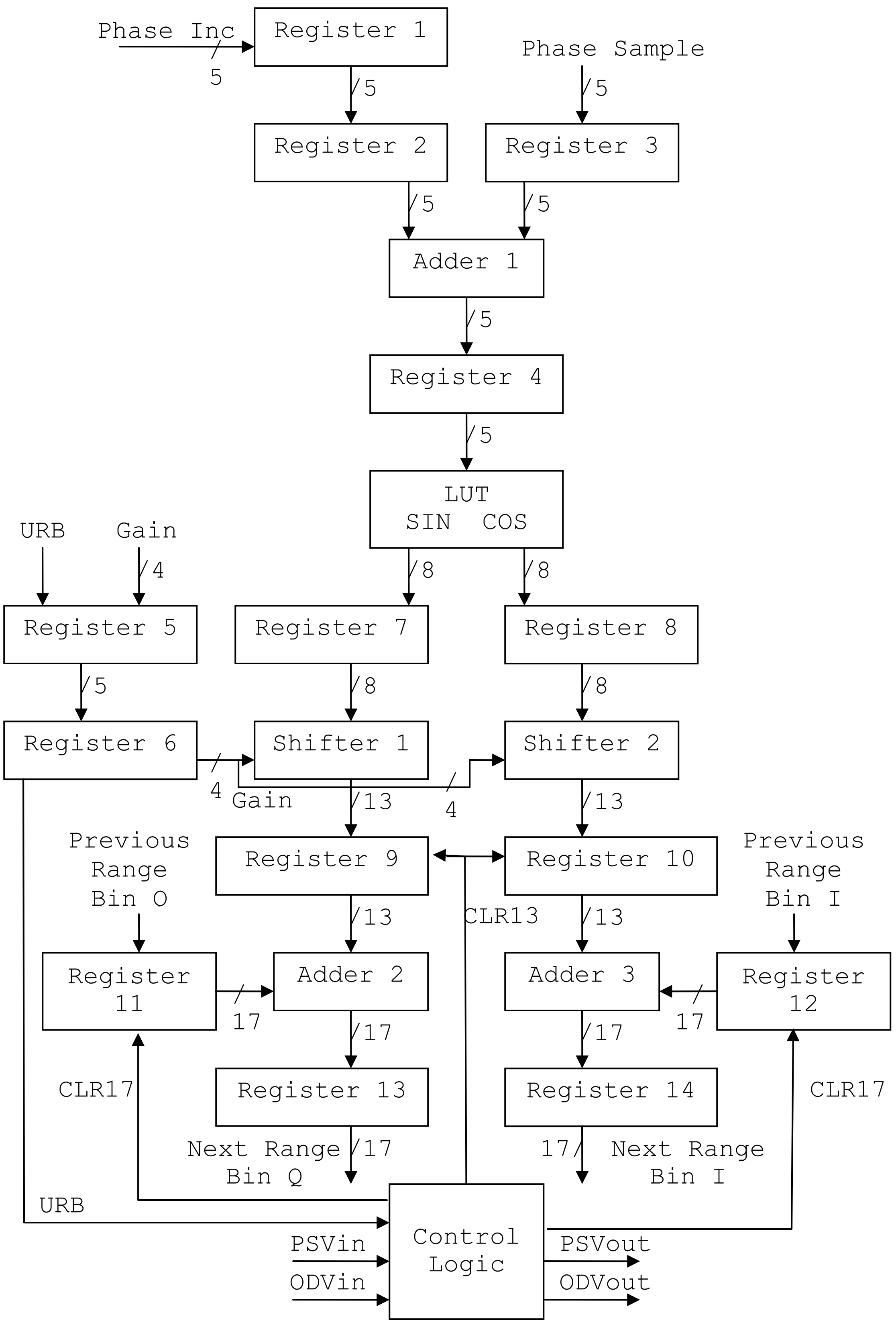

Figure 2. False Target Radar Image Chip Signal Flow

The basic steps of the algorithm are performed as follows:

1. The phase sample enters into register 3.

2. The phase rotation value enters into register 1, is then loaded into register 2, and is then added to the phase sample at adder 1. The results are then loaded into register 4.

3. The contents of register 4 enter the lookup table (LUT) and Sine and Cosine results are calculated. The remainder of the pipeline is split into two identical portions for each data result. The following steps outline the path for the Sine result.

4. The gain value enters at register 5, is then loaded to register 6, and proceeds to shifter 1 where it controls how the contents of register 7 are shifted before they proceed to register 9. This accomplishes modulo-2 multiplication.

5. The result from a preceding range bin enters at register 11 and is added to the contents of register 9 in adder 2 before proceeding to register 13.

6. The contents of register 13 are now available as output Q if this is the last range bin in the series or are sent to register 11 of a following range bin.

The control logic block receives signals URB (use range bin), PSVin (phase sample valid input), and ODVin (output data valid input). These signals are used to create the CLR13 (clear 13-bit register), CLR17 (clear 17-bit register), PSVout (phase sample valid output), and ODVout (output data valid output).

The internal design of the control logic is shown in Figure 3. The CLR13 and CLR17 signals clear the register contents at the appropriate time in the pipelines when they do not contain valid data. This occurs during pipeline startup and shutdown. The PSVout signal shows the DRFM signal is valid. The ODVout signal shows that outputs Q and I contain valid data. The URB signal allows the operator to disable a range bin. Figure 4 shows the signal flow when four range bins are connected together.

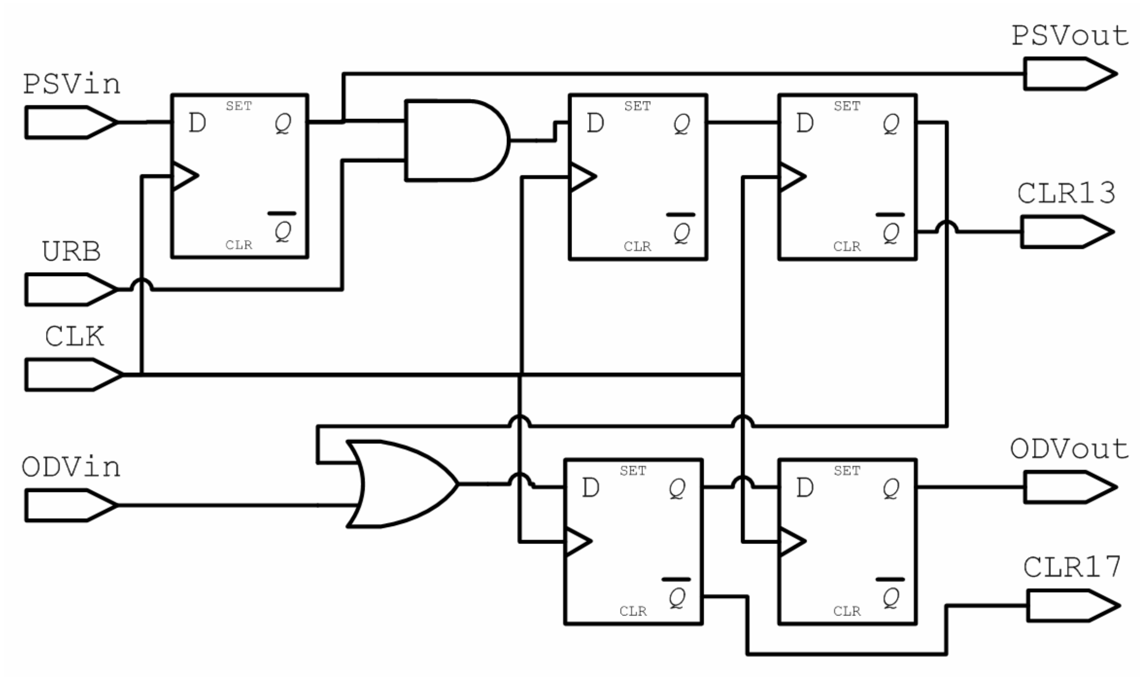

Figure 3. Internal Design of the Control Logic

D. FALSE-TARGET RADAR-IMAGING PROGRAM DESIGN

A C language program simulating the chip, originally written by Professor Douglas Fouts, was used as a standard for comparing the accuracy of results throughout the research. It performs the same arithmetic calculations as the chip but uses nested-loop iterative structures instead of pipelines. While the chip requires a separate pipeline for each range bin, the program simply adds additional length to the appropriate arrays, trading off memory utilization for computational logic. Table 1 shows how the results of each of four range bins with an input of *N* samples are placed into the two dimensional array created by the program. Each row of the table is then summed up to produce the false target radar signal results.

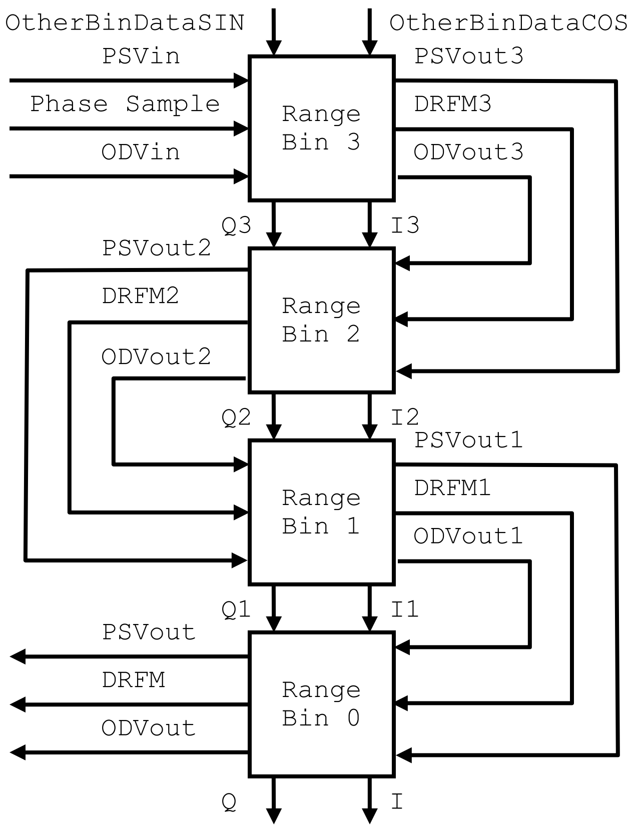

Figure 4. Signal Flow for Four Cascaded Range Bins

Bin 0	Bin 1	Bin 2	Bin 3
Sample 1 Results	0	0	0
Sample 2 Results	Sample 1 Results	0	0
Sample 3 Results	Sample 2 Results	Sample 1 Results	0
Sample 4 Results	Sample 3 Results	Sample 2 Results	Sample 1 Results
…	…	…	…
Sample *N* Results	Sample *N*-1 Results	Sample *N*-2 Results	Sample *N*-3 Results
0	Sample *N* Results	Sample *N*-1 Results	Sample *N*-2 Results
0	0	Sample *N* Results	Sample *N*-1 Results
0	0	0	Sample *N* Results

Table 1. False-Target Radar-Imaging Program Example Using Four Range Bins

The program was used as both a trusted source for results to test the research against as well as used in the timing comparisons in Chapter VI.

E. BENEFIT AND COST OF PIPELINING ALGORITHMS

The implementation of the false-radar imaging algorithm has each range bin's function split over 5 pipeline stages. The output of each range bin is then chained into the next in the final pipeline stage. This section will discuss the benefits and costs of pipelining.

1. Background

Pipelining allows work that involves repetition to be performed in parallel, reducing the total completion time. It is most effective when the work can be separated into tasks of equal length. However, in reality, work seldom splits into subtasks of equal length. Since the completion time for each stage varies with the task it performs, some delay is required at the end of each stage, so that slower stages have time to complete their task before receiving new work. Each of these tasks then becomes a pipeline stage where a portion of the total job is performed for each item passing through the pipeline. Each stage receives a result from the previous stage, performs its task, and then passes the new result to the next stage.

For pipelining on a chip that involves arithmetic functions performed by a computer, each pipeline stage consists of the logic to perform the subtask and a register which stores the result until the next stage is ready for it. A clock performs coordination of the entire pipeline by allowing the registers to all fill at the same time. The clock can only run as fast as the slowest pipeline stage. The speed of each stage includes the time to complete the function as well as the delay time caused by the register itself.

2. Pipeline Benefits

Pipelines have two major benefits. First, as previously discussed, they allow multiple repetitive tasks to be performed in parallel which reduces the total completion time. Second, pipelines can allow computer logic to run at a higher clock speed since they need not require more complex logic functions that require more time to complete. When the total time required is longer than the desired clock speed, the function can be broken into smaller pieces and placed in a pipeline.

3. Pipeline Costs

Since functions can rarely be broken into equal subtasks and pipeline registers introduce some delay, the total time to complete a function on a single data item by a pipeline will always be longer than the time to do it on that data item without a pipeline. However, when the function must be repeated the pipeline can become efficient. The point where the pipeline becomes efficient can begin no sooner than when the number of repetitions exceeds the number of stages. Also, if the function is not split up into fairly equal subtasks and/or the register delay is high, the onset of pipeline efficiency can be delayed even further. Pipeline speedup of a k-stage pipelined function over an equivalent non-pipelined function is defined as [3]:

$S_k = \frac{T_1}{T_k} = \frac{nk}{k+(n-1)}$, where n is the number of repetitions.

The implementation of the false-radar imaging-algorithm has each range bin's function split over five pipeline stages. The output of each range bin is then chained into the next in the final pipeline stage. This creates a complete pipeline length of N+4, where N is the number of range bins implemented. Figure 5 shows the pipeline speedup for the number of range bins implemented in this research, using N for *k* and number of samples for *n*.

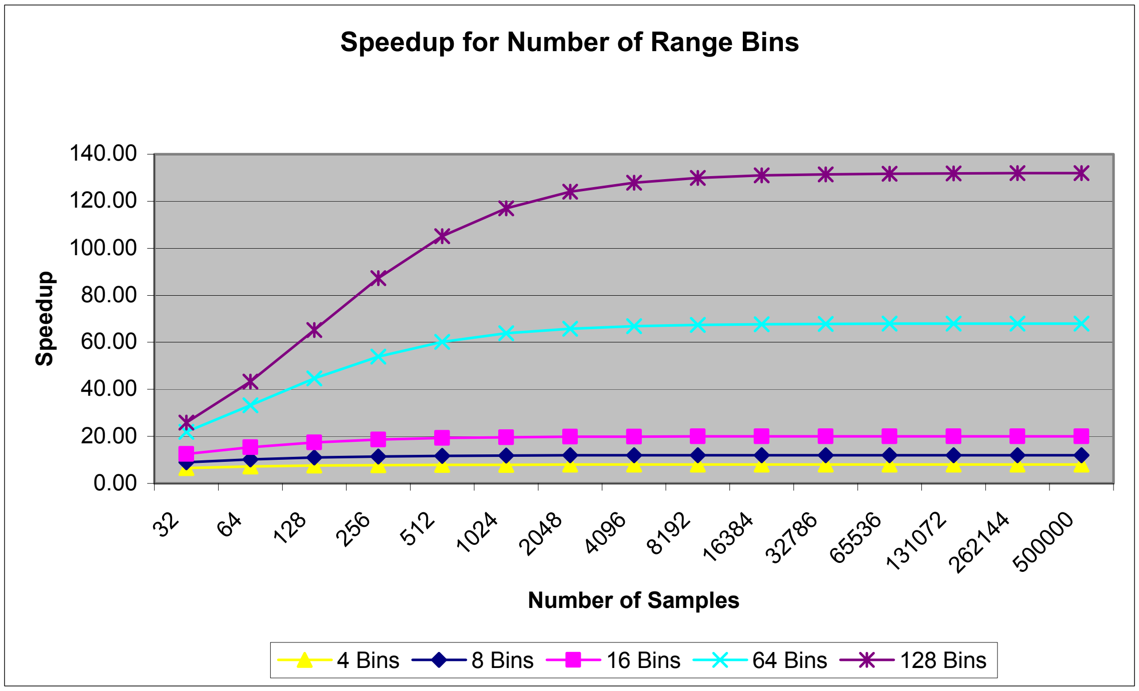

Figure 5. Speedup vs. Number of Range Bins.

The plots shows speedup increases as the number of samples is increased. For *n*>>*k*, maximum speedup of *k* is achieved, meaning *k* tasks are efficiently performed at the same time. For low *n*, the pipeline length is too long to be efficient. This phenomenon can be viewed in the data presented in Chapter VI.

F. SUMMARY OF NEW RESEARCH

Previous work implemented and tested four range bins. For simplicity, the previous design passed all the range bin programming data with each new radar sample. However, this was inefficient since the programming data does not change over a given set of radar samples. For each 5-bit sample, 40 bits of repetitive programming data was reentered, (10 per range bin). The interface to the MAP consists of 6 banks of 64-bits, of which at least one must be designated for output, leaving at most 320 bits for input. With the expansion to 16 range bins, half of the input bandwidth would have been consumed by unnecessary, repetitive programming data and further expansion beyond 16 bins would be impossible within the provided bandwidth. The current research implements 4-, 8-, 16-, 64- and 128-range bin versions using a new design which allows the programming data to be sent to the macro only once for each set of radar samples.

III. SRC-6E ARCHITECTURE AND SOFTWARE ENVIRONMENT

A. INTRODUCTION

This chapter provides an overview of the hardware, software, and documentation, of the SRC-6E reconfigurable computing system.

B. SRC-6E HARDWARE OVERVIEW

The SRC-6E computer consists of two independent Linux computers (labeled A and B) and a MAP board (see Figure 6).

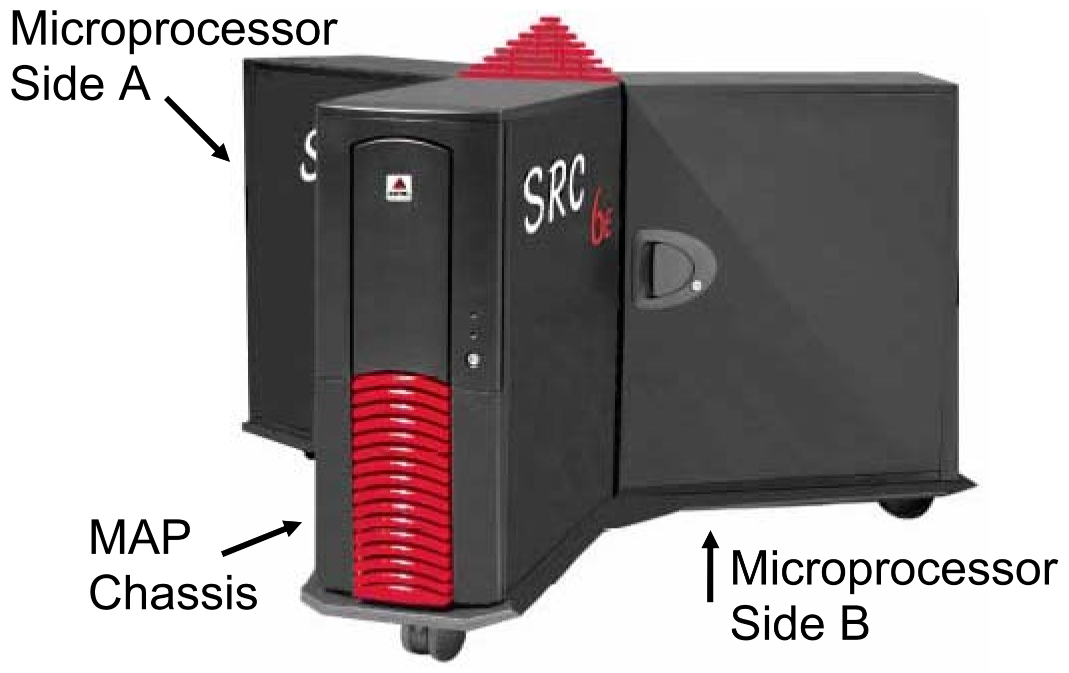

Figure 6. SRC-6E System Diagram (After Ref. 4.)

MAP, a registered trademark of SRC Computers, Inc., is the name of the custom reconfigurable hardware. Each independent Linux computer contains a motherboard holding dual Intel P3 Xeon 1-GHz processors, 1.5 gigabytes of memory, and a SNAP interface card. The SNAP card is a custom interface card which plugs into a motherboard DIMM memory slot and provides connections to the MAP board which is located in the MAP Chassis. A single MAP board consists of two independent MAPs. A block diagram of a single MAP is shown in Figure 7. A MAP consists of three Xilinx Virtex-II-series XC2V6000 FPGAs and 24 megabytes of memory (labeled OBM in Figure 7).

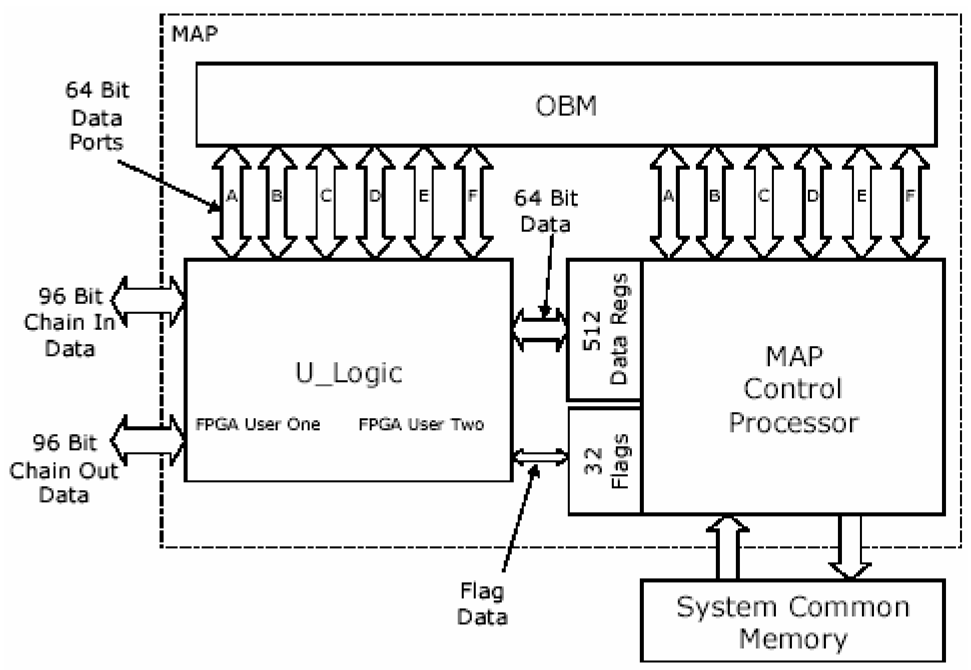

Figure 7. MAP Interface Block Diagram (From Ref. 4.)

One of the FPGAs is reserved for control logic while the other two, available for user programs, are user logic. The OBM memory is split into six equal banks, labeled A through F, of 4 megabytes each. The user FPGAs are connected to a fixed 100-MHz clock which seems overly inflexible. According to Xilinx product specification sheets, the Virtex-II-series FPGAs can run at clock speeds as low as 1 MHz and upwards of 400 MHz [5]. Programmer control of the clock speed on the SRC-6E would make the system more flexi ble. Each MAP also has a chain port which can be used for direct I/O to the user logic, but was not used during this research.

C. SOFTWARE ENVIRONMENT

1. Operating System

The operating system for the SRC-6E is Red Hat Linux, and it has been augmented with custom drivers and libraries to support the MAP and SNAP hardware. The built-in graphical text editor in Linux is called GEdit. Programmers experienced with UNIX can use the standard line-text editors such as VI if they choose. Both contain the minimal functionality required of a text editor to write the required files for the SRC-6E.

2. Programming Environment

The programming environment for the SRC-6E is called Carte. Carte allows a user to write code in a high level language, either C or Fortran, that directly targets the user programmable FPGAs in the MAP. In addition, users can write their own macros using the hardware definition languages Verilog and/or VHDL. At compile time, all user code and macros are linked together into a single executable file. Carte compiles for the Intel microprocessors as well as the MAP, and for both Fortran and C. Synplify Pro software by Synplicity, Inc. is used for FPGA place and routing. This program normally runs under Windows version but is executed in the Linux environment using a Windows emulator called Wine.

Since Carte relies on the built-in Linux editors, the SRC-6E programming environment does not have any of the modern features a programmer expects from editors available in products such as Microsoft’s Visual C++ or Borland’s J-

Builder. Lack of syntax and semantic error checking in the programming environment is a serious drawback when using this system. Some error messages are produced at compile time, but they are cryptic at best, especially for someone not familiar with the Linux environment. There are several file types which must interact during the compile process, as will be discussed in Chapter V. The intricate details of these files can be quite confusing and it is often difficult to identify which file contains the problem based on the error messages given at compile time. Rudimentary checking of these files within a custom editor would greatly improve the entire programming process.

D. MAJOR DOCUMENTATION

Several documents come with the SRC-6E to aid in its programming.

1. SRC-6E C Programming Environment Guide

Driver code must be developed to create the interface to the user logic. This document describes how to write this code using the C language [6].

2. SRC-6E Fortran Programming Environment Guide

Similar to the C Programming Environment Guide, this document describes how to write similar code using the Fortran language [7].

3. SRC-6E MAP Hardware Guide

This document contains hardware implementation specifics of the MAP which are well below the level required for users to successfully program the SRC-6E [4].

4. SRC-6E MAP Macro Developers Guide

This document discusses general information on the use of the Macro Data Sheet Library, including naming conventions, interfaces, fanout and combinatorial delays [8].

5. Macro Data Sheet Library

The library contains data sheets for all macros developed by SRC for the SRC-6E. A list of all currently supported macros is available in a technical note, Ref. 9. The macros can be used like regular function calls in the chosen programming environment language (C or Fortran). The macros include all basic math and logic functions currently supported by the environment. There are also several support macros which include, among others, various macros for combining and splitting data structures.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. DEVELOPMENT AND TESTING IN VHDL WITH ALDEC ACTIVE-HDL 5.2

A. INTRODUCTION

This chapter describes our development of the false-target radar-imaging macros in VHDL before they were ported to the SRC-6E environment. The final code used to implement 4 range bins in the previous research was a starting point and is hereafter referred to as the "old macro". Development of the macros was performed in a Windows XP environment using Aldec Active-HDL 5.2 software.

B. MODIFICATIONS TO PREVIOUS MACRO

The old macro used two 64-bit inputs and a single 64-bit output. Assuming that a reduction in overhead would improve their efficiency, the new macros were developed to use only a single 64-bit input and output. Other modifications included addition of the PRB and UNP signals and internal connections of the URB signal within the range bins. 4-, 8-, 16-, 64- and 128-range bin versions were created. The code for this section can be viewed in Appendix A.

1. Overhead Reduction

To reduce the input bandwidth, it was necessary to send in the programming data only once for each set of radar samples. This was accomplished by complete redesign of the interface with implementation of the PRB and UNP signals. These signals were present in the original chip design but were not implemented in the old macro.

2. PRB Signal

The PRB signal, which stands for "Program Range Bin," was originally intended to signal a specific range bin to accept new programming. As implemented in this interface, it signals the start of a programming sequence where four range bins are programmed per clock cycle until all are programmed. Thus for the sixteen bin model it takes 4 clock cycles to program all range bins. The PRB is delayed through a shift register which is appropriately sized for the number of range bins. Each group of four range bins beyond the first requires a bit. A three-bit shift register is used for the 16 range-bin macro, one bit for the 8 range-bin macro, and no shift register is necessary for the 4 range-bin macro. All range bins are sent programming data at the same time, but the delayed PRB signal causes only the appropriate group of range bins to load the data.

3. UNP Signal

The UNP signal, which stands for "Use New Programming," was originally intended to signal each range bin one at a time to begin using its new programming. The UNP signal was not implemented in the old macro. As implemented in the new macros, it signals all range bins to simultaneously use their new programming. The UNP signal is normally low. Taking the UNP signal high for one clock signals the macro to use the new programming data previously loaded by a PRB programming sequence.

4. URB Signal

The old macro used a URB signal which stands for "Use Range Bin." Intended as a single bit for each range bin to signal the control logic as to whether or not to use a particular range bin, this signal is programmed during the PRB programming sequence. When low, the output of a range bin is not included in the overall output. The old macro assumed that all four range bins would be used and had this signal tied high. The new macro designs have the URB signal internally connected to allow its full intended use.

5. Experiments with Different Numbers of Range Bins

The code for four range bins was created by instancing the original single range bin code and creating a new, more efficient interface. The signal flow of four range bins is shown in Figure 4. The code was verified by comparing the output to the C program run on the same data set. This model was created to draw a timing comparison between the old macro and the new design. This will determine if the reduction in the interface has any affect on the overall timing.

A system with eight range bins was created by instancing eight of the single range bins previously tested. This model was developed as an intermediate size for collection of timing data.

A system with sixteen range bins was created by instancing sixteen of the single range bins with an appropriate interface. This model was the original goal of the new research. Through reduction of the interface overhead by 1/3 and increase of the number of pipelines by a factor of 4, it was hoped that the SRC-6E would show an improvement in timing as compared to the other benchmarks.

A system with 64 range bins was created by instancing 64 of the single range bins with an appropriate interface. This model was developed because of the unexpectedly low chip usage by the previous models which will be discussed in the next chapter.

A system with 128 range bins was created by instancing 128 of the single range bins with an appropriate interface. This model was developed after the 256 range bin model would not fit in the user logic area.

THIS PAGE INTENTIONALLY LEFT BLANK

V. PORTING THE VHDL CODE TO THE SRC-6E

A. INTRODUCTION

This chapter discusses the porting of the VHDL code to the SRC-6E and the required support files. Also discussed are changes that were required to the original code to make it compatible with the SRC-6E.

B. THE SRC-6E FILE TYPES

The process of writing code to target the user logic requires several file types. To import a user macro from either VHDL or Verilog, five files must be created: .info, .box, .mc, .c, and the makefile. Using only the last three, one can write code that targets the user logic without a user-defined macro. Examples of these file types can be viewed in Appendix B, which contains the final versions of all the files used.

1. .info

This file type is required whenever a user macro is used. It contains the following information:

- Macro name
- Macro type – stateful, external, and pipelined
- Latency – a number stating how many clock cycles before valid output is generated by the macro.
- List of inputs and outputs

2. .box

This is also required only for a user-defined macro. It is a Verilog style description of the input and output variables of the macro. The Verilog description is necessary for both VHDL and Verilog macros.

3. .mc

This file type is for C code written to target the user logic. All code in this file will be implemented in hardware along with the user macro. Using this file type, it is possible to write code for the hardware using only the high-level language C without using any user-defined macros.

4. .c

This file type is for regular C code which provides the interface between the operating system and the hardware code defined in the .mc file. Code implemented in this file is executed on the Xeon processors.

5. makefile

This file is used by the command "make" when all the files are compiled and linked. It contains all of the file names and paths used, as well as the desired final executable name. Compiler flags and options can also be stated in this file.

6. .vhd

This file type is for VHDL macro files. In general, it is safest to merge multiple files into one. However, it is possible to build a program with separate files as long as they are listed in the proper order in the makefile. The compiler appears to be single-pass so the files must be in the order they are used, with the lowest-order file listed first.

7. Other Types

Two other file types can be used by users programming the user logic: .f, which is a Fortran file, and .v, which is a Verilog file. These file types were not used during this research.

C. CODE DEVELOPMENT

Porting of the macro code began with creation of the required support files previously mentioned. Modifying the support files from the old macro was fairly simple. Only the .c interface file required a significant rewrite to match up with the new macro interface. The code went through three major versions and several revisions.

1. Version 1.0

The sixteen-range-bin VHDL code was imported to the SRC-6E. The required support files were generated by modifying the old macro files to support the new design; this proved trivial compared to the long trial-and-error process reported in the previous research. However, during the make process, the SRC-6E reported a final clock speed of 78.5 MHz, well below the required minimum of 100 MHz. Without surprise due to the timing failure the generated executable did not produce the proper output. This version implemented the programming sequence using a finite-state machine with four states using only the UNP signal to reset the state to zero. While this version created the proper output with the Aldec software, it generated too much delay when implemented in hardware on the SRC-6E.

2. Version 1.1

An eight-range-bin version was created to reduce the macro complexity for troubleshooting. After verification in Aldec of the proper output it was compiled on the SRC-6E. This version reported a final clock speed of 84.6 MHz and also generated incorrect output.

3. Version 1.2

A four-range-bin version was created to further reduce the macro complexity for troubleshooting. After verification in Aldec of the proper output it was compiled on the SRC-6E. This version reported a final clock speed of 85.1 MHz and also generated faulty output. Since the old macro had generated proper four-range-bins output on the SRC-6E it was determined that the finite state machine method would not work on the SRC-6E, presumably due to the style of programming it and not failure of the SRC-6E. After discussion with SRC technical support, the need for a simpler more direct way of implementing the interface became apparent.

4. Version 2.0

Review of the actual functionality required by the state machine showed that the same tasks could be performed with simpler components. Since the states only changed in numerical order, one per clock, the same function could be performed with a counter. Since this would require a significant redesign, a review of the original chip design was conducted to see if the macro implementation could be brought closer to the original design. During the review it was discovered that if the original PRB and UNP signals were implemented properly, a simple shift register could perform all of the necessary functions to direct the programming input as required. The new design was implemented and verified in Aldec to produce the proper output. After porting the new macro to the SRC-6E and modifying the .c file to support the new interface, the SRC-6E produced the proper sixteen-range-bin output for the first time. Following the same basic design, new 8- and 4-range bin macros were rapidly created and verified.

5. Version 3.0

After unexpectedly low FPGA usage for the 8- and 16-bin macros, only 12% and 16%, a 64-bin macro was implemented. The new version demonstrated an FPGA usage of 38%; a 128-bin version demonstrated a usage of 68%. A 256-bin version was attempted but it would not fit within the user logic.

D. SYNTHESIZABLITY

As noted in the original research, synthesizability was a problem during the design process. Synthesizability is a style of hardware description language programming which allows the available layout tools to properly convert the code for hardware implementation on an FPGA. During the design of the code, the Aldec software was only used to simulate the VHDL code. Therefore, it only tested the code for functionality and did not consider if the code could actually be implemented in hardware. Problems with the state machine design resulted in synthesizability problems which were resolved with a redesign of the interface. The root cause of this was inexperience with both the VHDL language and the concept of synthesizability.

E. FPGA UTILIZATION

As previously noted, FPGA utilization was much lower than expected. Based on the 4-bin’s 10% usage, slightly over 20% and 40% were expected for the 8 and 16 bin models, based on linear scaling of the number of additional pipelines implemented by the two larger models. The observed 12% and 16% usages show that the Xilinx software is performing some unexpected and very effective optimization of the VHDL code which causes the actual usage to be much lower than the original estimations. The 64- and 128-bin versions showed usage of 38% and 68%. A 256-bin version was attempted but after three days of compiling failed with a usage of 99% with a little over 500 signals left to route. Thus 256-bins almost fit within the user logic area. However, it should be noted that this version had multiple timing failures resulting in a clock speed of only 32.8 MHz which would not have produced correct results. Figure 8 is a plot of actual FGPA usage vs. the number of bins. Two best fit plots, one exponential and one second order polynomial are added to predict the FPGA usage for macros with other numbers of bins. The polynomial curve seems to most closely match the actual data.

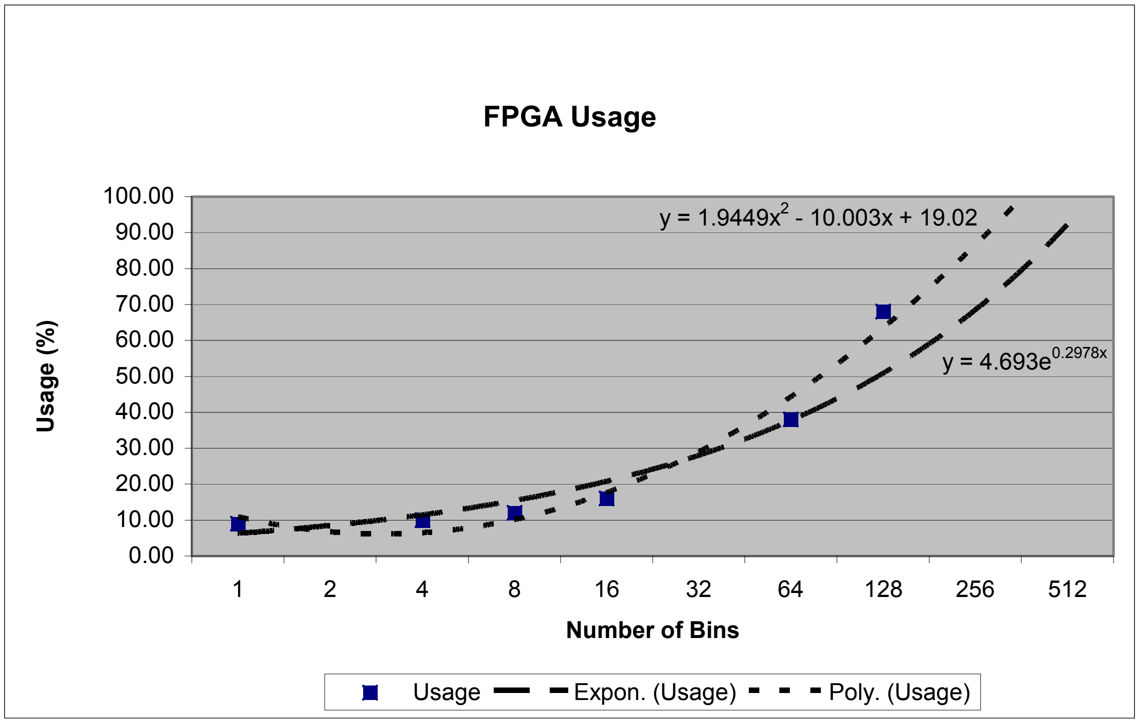

Figure 8. FPGA Usage

VI. PERFORMANCE RESULTS

A. INTRODUCTION

This chapter discusses the benchmarks, methods used for collection of data, and data analysis.

B. BENCHMARK TEST PLATFORMS

1. C Program Executed on a Windows-based Machine

A C program was compiled and executed on an off-the-shelf 3-GHz Pentium 4 processor system with 2 gigabytes of RAM running the Windows XP Professional operating system. The primary reason for this benchmark was to draw a comparison for cost effectiveness between the high-cost special-purpose SRC-6E system and a modern off-the-shelf general-purpose computer. The same computer that was used for the previous research was used to collect the new data.

2. C Program Executed on the SRC-6E

The same C program was also compiled and run directly on the SRC-6E without using any of the custom hardware. The data collected was based on the Linux operating system running on a 1-GHz Xeon 3 processor with 1.5 gigabytes of RAM. Although the system contains dual processors, only one thread was created while running the code and therefore it is believed that only one processor was used during the test. The primary reason for this benchmark was to test if the algorithm itself is suitable for implementation on the user logic.

3. VHDL Code on the SRC-6E MAP

The VHDL user macro and support files (shown in Appendix B) were also implemented and executed on the SRC-6E MAP. Two timing data results were collected from each of the runs, the total run time for the entire execution and the time of execution on the MAP only. The two timing data results compare overhead time to actual execution on the MAP.

C. TIMING DATA COLLECTION METHOD

The data collection method used in this research was identical to the previous work to allow for direct comparison of the results. The input data sets were composed to represent a stream of intercepted radar samples. Each data item consists of two hexadecimal characters representing a five-bit intercepted radar sample. The 32-sample-size data set is shown in Appendix B, which represents the decimal numbers 0 to 31 in order. All other-sized sample sets were created by duplicating and repeating the same 32 samples in order. By doubling each previous sample set size the following set sizes were created: 32, 64, 128, 256, 512, 1,024, 2,048, 4,096, 8,192, 16,284, 32,786, 65,536, 131,072, and 262,144. The final set size of 500,000 was chosen as a convenient, large value that was close to the upper array size restriction allowed by the four megabytes of memory per bank on the SRC-6E.

Data from all test platforms were collected in order of increasing input set size. The timing data was collected by running five consecutive runs of each input data set on each of the three benchmark platforms. The data for the Windows XP system were collected after a fresh reboot with all unnecessary programs closed. As noted in the previous research, observation of the system usage during execution of the code again showed that the processor and memory were not fully used, never exceeding 54% usage during the 16-range-bin tests. The reasons why the processor did not appear to be fully used and the methods Windows uses to measure performance are unknown. The SRC-6E system data were collected by running the executables on side A when no other users were using the system.

D. TIMING DATA ANALYSIS

1. Methods

The timing data are displayed in two types of graphs for each range bin size. The first is the average total time each test platform takes for each data set. The average is taken of the five data points for each input set size. The second is the average time per sample for each input set. First, the average is taken over the five data points and then it is divided by the input set size. All graphs are connected with straight line approximations between data points.

2. New Interface Timing Results

Figure 9 shows the SRC MAP call average total time vs. input set size for each of the implemented range bin sizes.

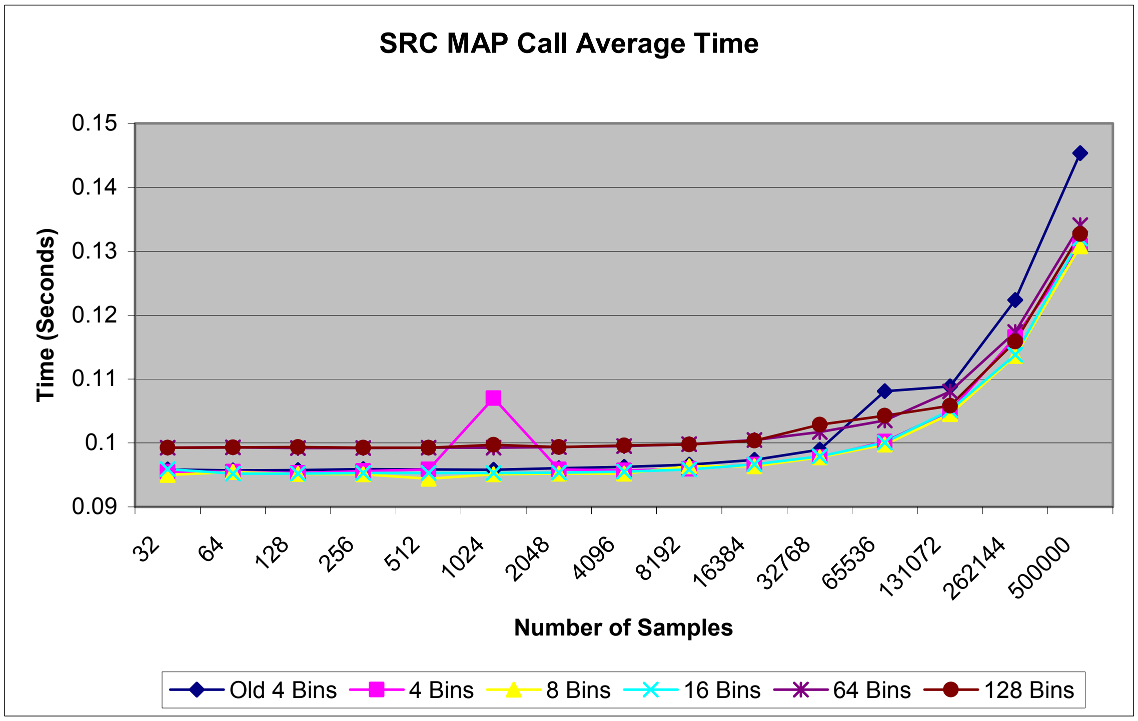

Figure 9. Comparison of MAP Call Average Time

The Old 4 Bins plot shows that the new macro interface is more efficient for large input set sizes. The 4 Bins plot shows a spike at 1024 samples, caused by a single unusually large time result in the data. The reason for this one slow result is unknown.

3. 4 Range Bin Timing Results

These results are very similar to those found in the previous research. Figure 10 shows the average total time for 4 range bins.

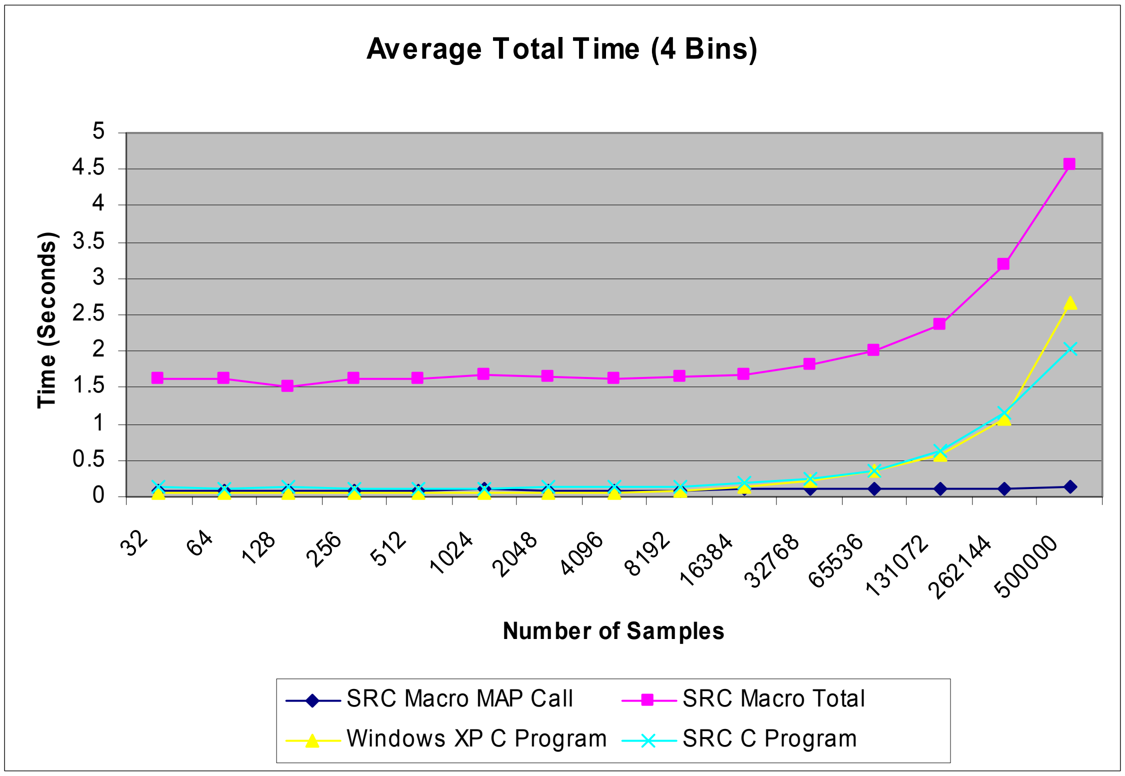

Figure 10. Average Total Time (4 Bins)

All four curves are fairly constant up to the 16,284 sample size. This result shows that, for small data set sizes, the overhead times inherent in the systems are much greater than the calculation times. We consider overhead to be all the data file read/write operations and memory accesses required to prepare the data for calculations. The SRC Macro MAP Call curve clearly shows the calculation time is insignificant compared to the total processing time.

The SRC Macro MAP Call curve also shows the MAP execution time is extremely fast, even for very large data set sizes. However, the SRC Macro Total curve shows the total execution time for the VHDL macro takes considerably longer. The extra time represents delays in the system to prepare and transfer the data in and out of the MAP which cause the SRC execution time to be longer for all input set sizes, initially by an order of magnitude.

As the input set size is increased, we see the curves begin to converge. Figure 11 shows a comparison of the average time per sample. Figure 12 shows the same data on a semi-log scale.

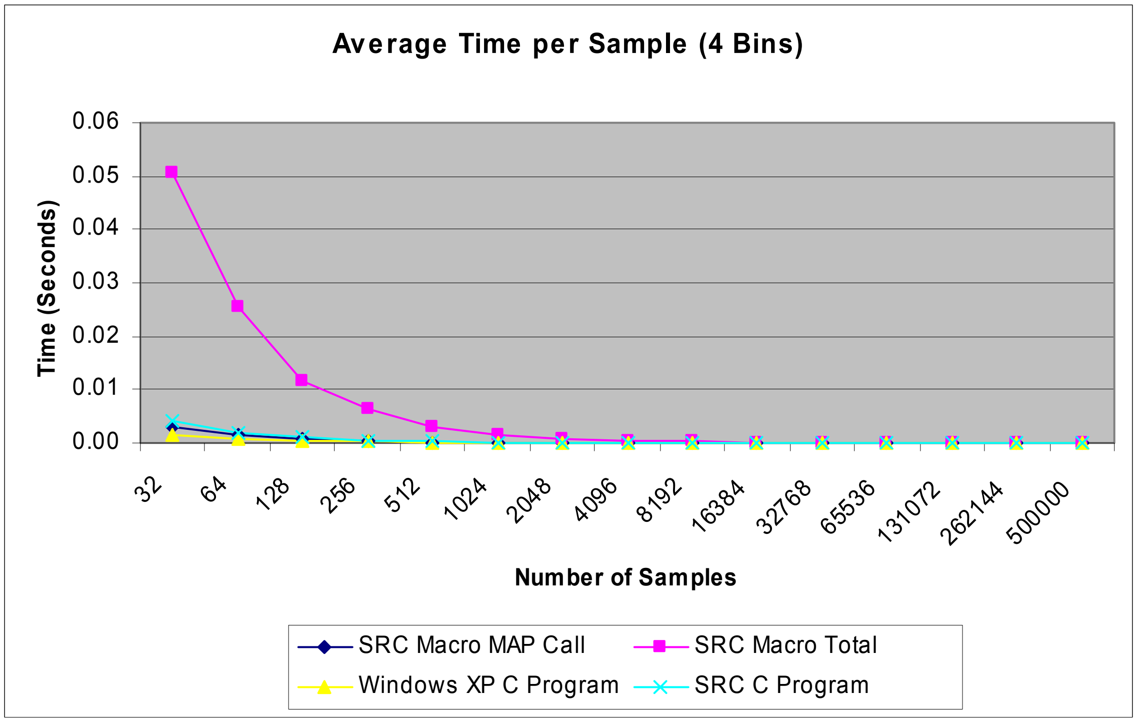

Figure 11. Comparison of Average Time per Sample

The overhead in the SRC Macro clearly dominates the graphs for smaller sample set sizes. However, for larger sample set sizes, the overhead time is amortized over the total time to be nearly insignificant. The SRC Macro Total time is approaching the other curves and presumably would eventually meet them if the sample set size could be further increased. However, this is not possible with the current macro design due to the memory design of the SRC-6E hardware.

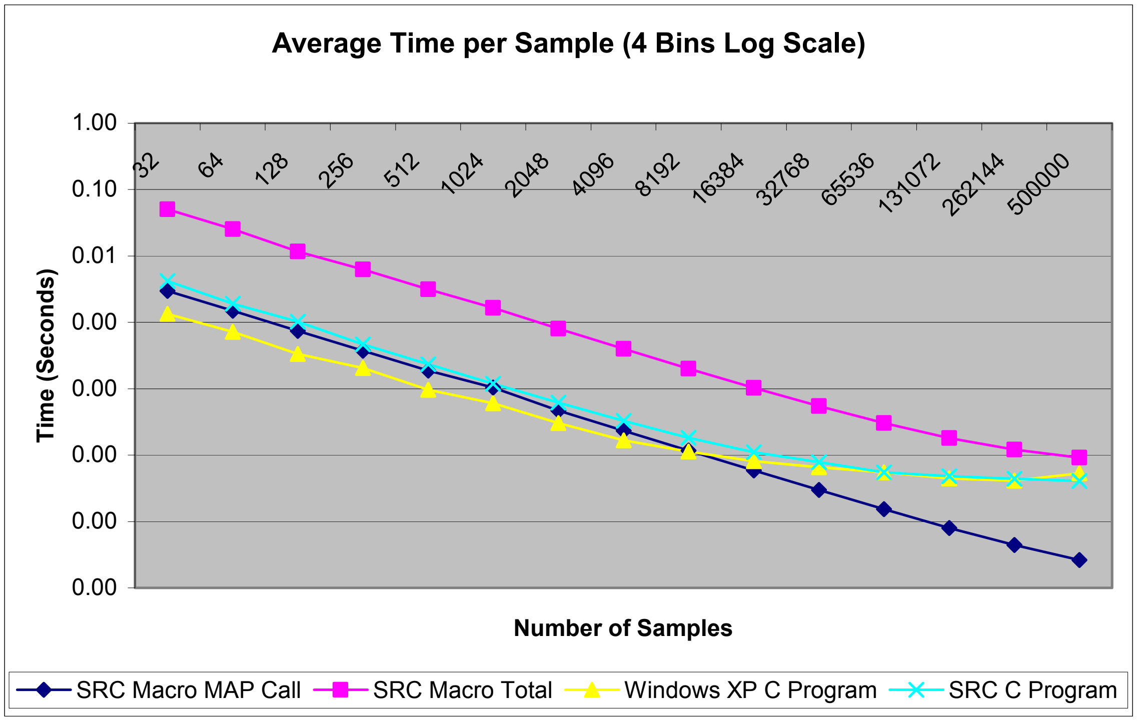

Figure 12. Average Time per Sample (4 Bins Semi-Log)

4. Timing Results for Larger Numbers of Range Bins

The results for 8 and 16 range bins are similar to 4 range bins. Figure 13 shows the average total time for 8 range bins. Figure 14 shows the average total time for 16 bins. The macros still lack enough parallelism to be effectively implemented on the SRC. However, it can be observed that the SRC macro time is remaining the same as it was for 4 range bins, while the C program implemented on the Windows XP and SRC are becoming slower.

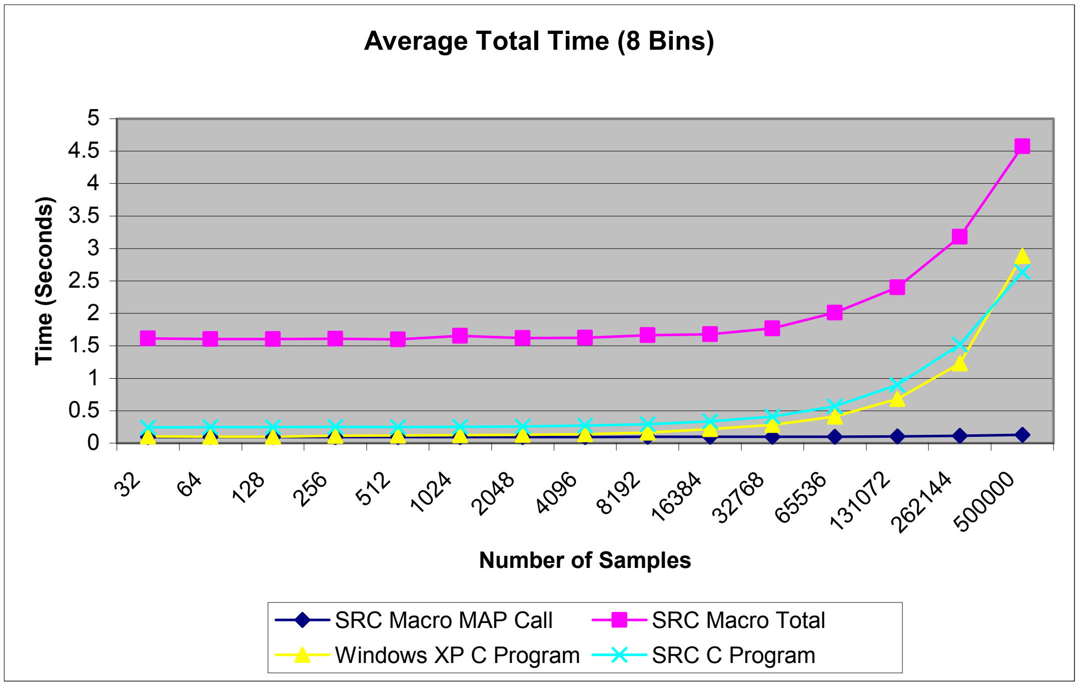

Figure 13. Average Total Time(8 Bins)

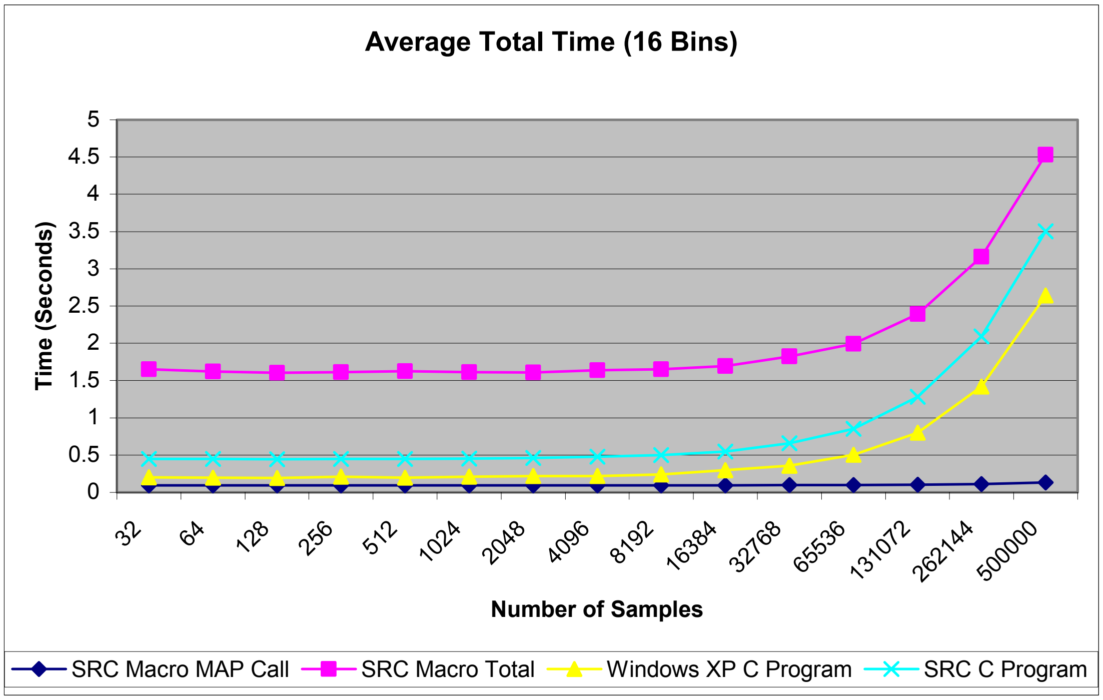

Figure 14. Average Total Time(16 Bins)

Figure 15 shows the average total time for 64 bins. For the first time, the SRC macro beats the SRC C program for all sample sizes, demonstrating that the macro implementation is more efficient than the Xeon processor running at 1 GHz. At 500,000 inputs, the SRC macro also meets the Windows XP C program running at 3 GHz. Therefore, 64 range bins marks the lower bound for range bin implementation that can be run efficiently on the SRC-6E.

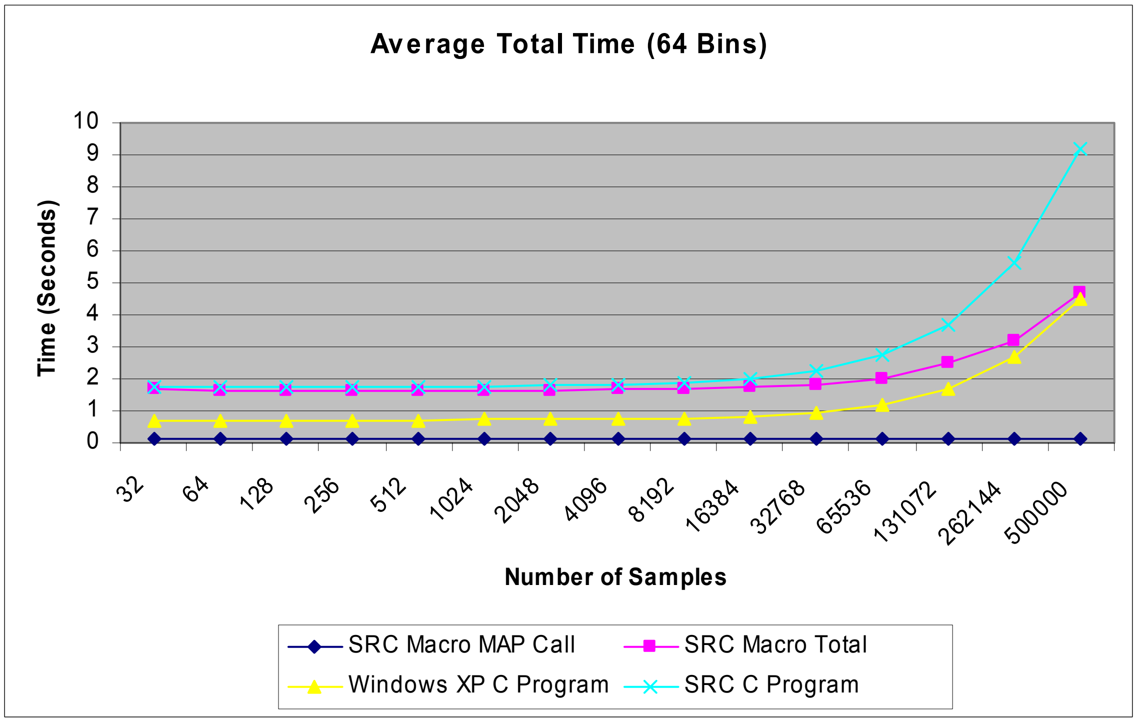

Figure 15. Average Total Time(64 Bins)

Figure 16 shows the average total time for 128 bins. The SRC macro beats the SRC C program again for all input sizes and by as much as an order of magnitude for large input sizes.

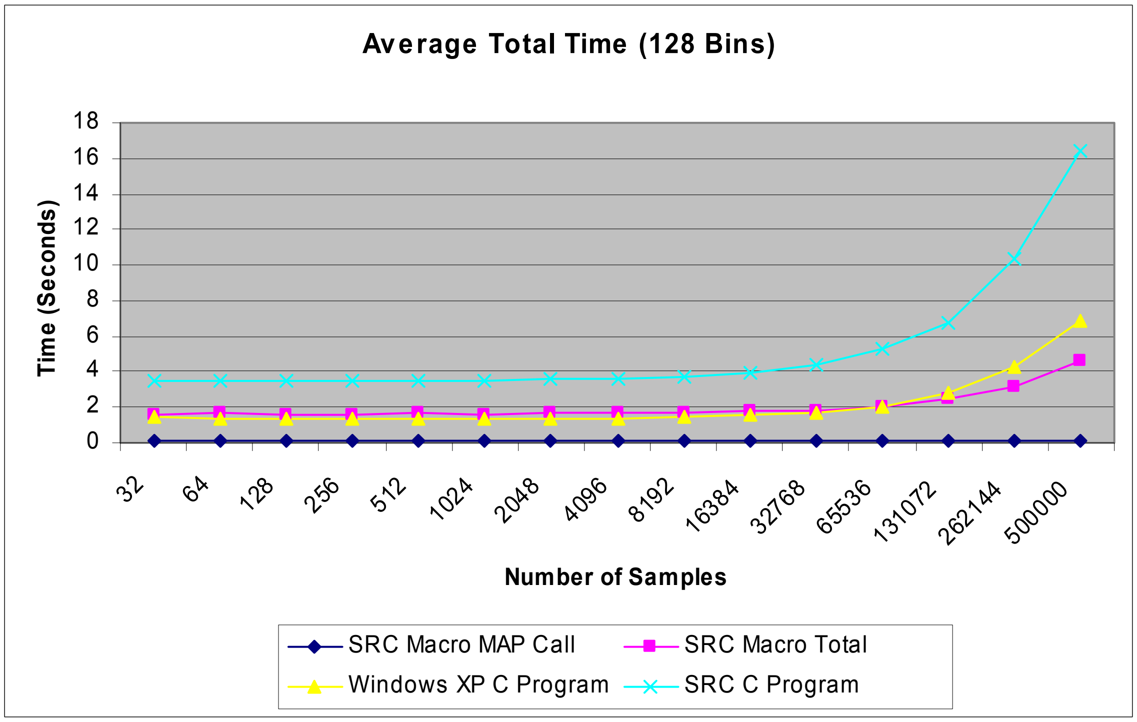

Figure 16. Average Total Time(128 Bins)

5. Contributors to Total Macro Execution Time

Further analysis of the macro timing results show three major contributors to the total execution time: I/O overhead, MAP overhead, and the MAP call. Figure 17 shows the percentage of total execution time of the three contributors for the 4 range bin model. The graphs for the larger range bin models are very similar showing only that the relative contributions of the three factors are independent of the number of range bins implemented. We will analyze each of the three contributors independently to determine a final equation of the form $T = I/O_{OH} + MAP_{OH} + MAP_{Call}$.

a. I/O Overhead

We define I/O overhead to include all read and write access to the data files as well as the preparation of the input data for the macro and the output data for presentation. Figure 18 shows I/O overhead is erratic for low numbers of inputs and independent of the number of range bins implemented for large input size.

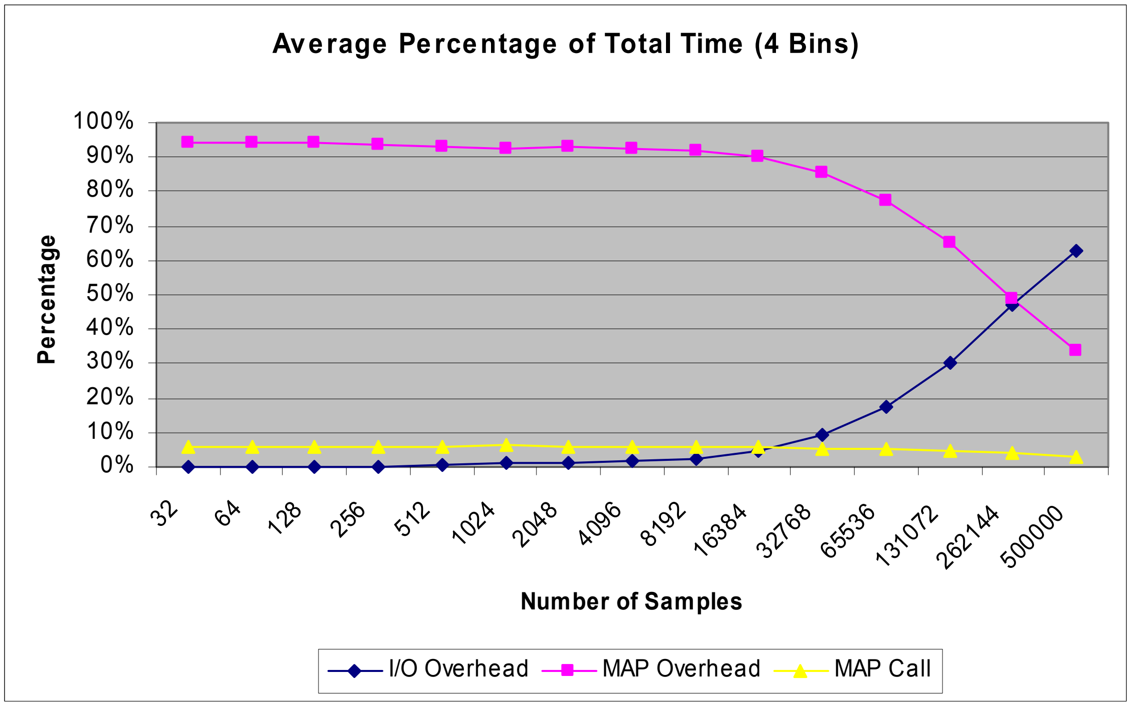

Figure 17. Average Percentage of Total Time(4 Bins)

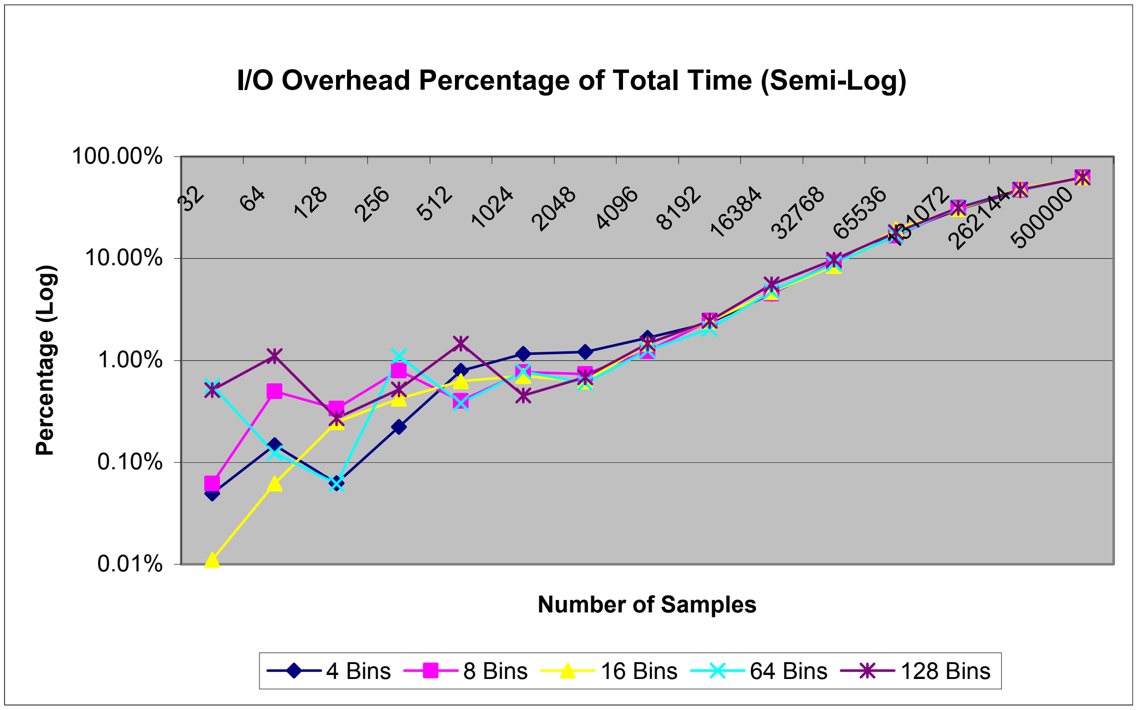

Figure 18. I/O Overhead Percentage of Total Time (Semi-Log)

Since overhead time is independent of the number of range bins, Figure 19 shows a plot of all I/O times for the 5 range bin models averaged together. A best-fit linear curve is included, which has a corresponding equation of $y = 5.77 \times 10^{-6} x - 2.47 \times 10^{-3}$. The zero crossing represents an insignificant 2 msec so will be ignored for simplicity. Using more appropriate variable names the final equation becomes $I/O_{OH} = 5.77 \times 10^{-6} S$, where S is the number of input samples.

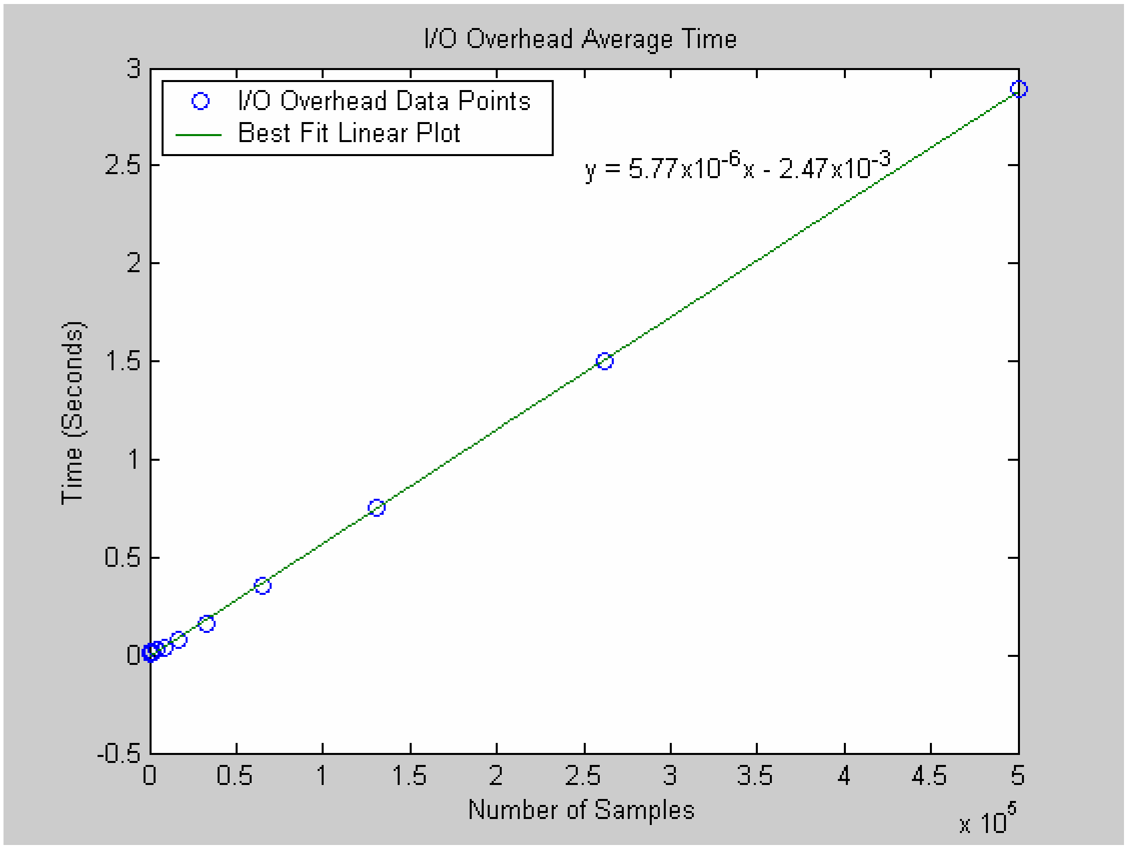

Figure 19. I/O Overhead Average Time

b. MAP Overhead

We define MAP overhead to be the sum of MAP allocation and deallocation times. Figure 20 shows that MAP overhead is independent of the number of input samples and the number of range bins implemented. MAP allocation time appears random within the range of 0.481 – 0.700 seconds. MAP deallocation time appears random within the range of 1.001 – 1.139 seconds. Relative to the other total time contributors, the range of the variation is quite small so an average value will be used as a constant in the final equation. Averages of all result sets yield 0.511 and 1.019 seconds, for allocation and deallocation respectively, yielding an average total of 1.53 seconds for the MAP overhead constant. Thus $MAP_{OH} = 1.53$.

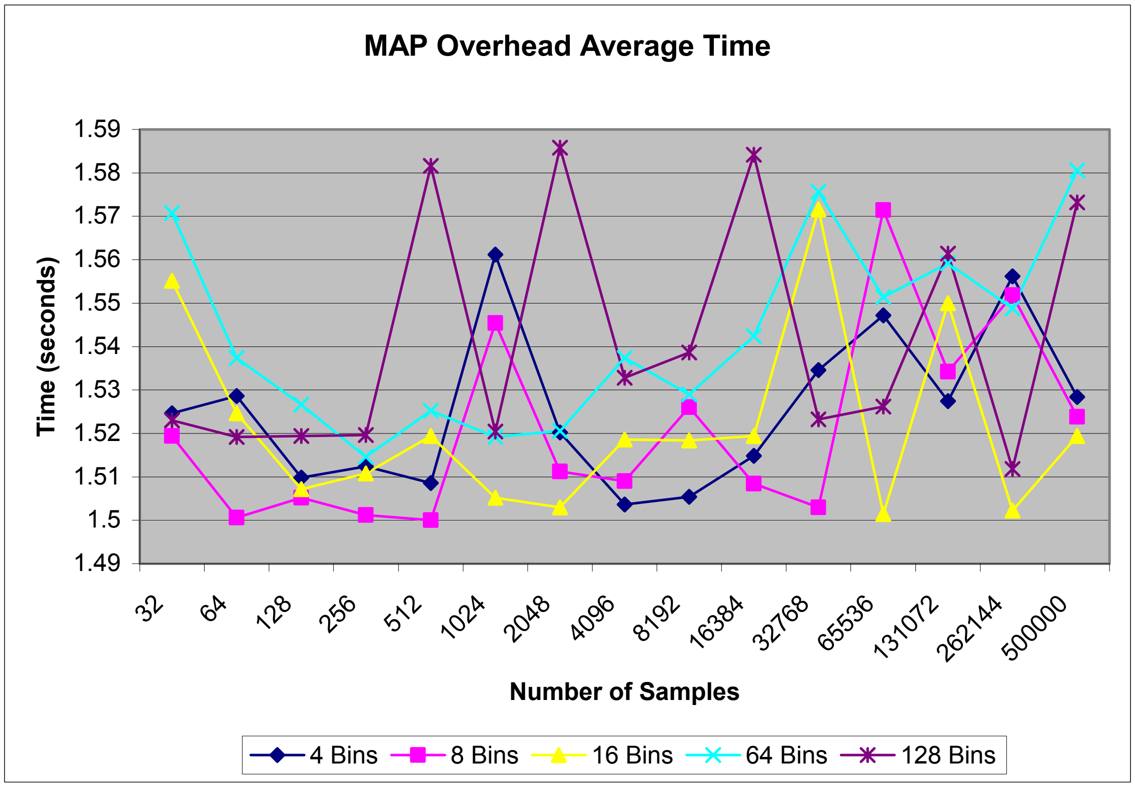

Figure 20. MAP Overhead Average Time

c. MAP Call

The MAP call time consists of the function call to the macro which is executed on the MAP. This includes the macro computation time as well as the input and output data memory transfers to and from the OBM to the system memory. Figure 9 previously showed that the MAP call has a very small dependence, below 10 msec, on the number of implemented range bins which is insignificant when compared to the total execution time. Figure 9 also shows a small dependence, about 35 msec, on the number of input samples, which is also insignificant. The MAP call has a minimum execution time of 95 msec. For the final equation, we will use the average of all MAP call data resulting in a final constant of 0.102 seconds. Thus $MAP_{Call} = 0.102$.

d. The Final Equation

Combining the three factors together yields the following equation where *S* is the number input samples:

$$T = I/O_{OH} + MAP_{OH} + MAP_{Call} = 5.77\times10^{-6}S + 1.53 + 0.102 = 5.77\times10^{-6}S + 1.632 .$$

Figure 17 shows that I/O overhead is insignificant below approximately 2000 input samples. In this range MAP_{OH} dominates the result yielding a constant value of approximately 1.6 seconds.

6. Conclusions

The design of the VHDL macro running on the SRC-6E suffers from excessive overhead which makes it less efficient than the C program which performs the same calculations for implementations below 64 range bins. Above 64 range bins the macro is very efficient, even beating a Windows XP computer running at a clock speed 30 times higher under certain conditions. Total macro execution time in seconds can be estimated as $T = 5.77\times10^{-6}S + 1.632$ where S is the number of input samples.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSIONS

A. DIFFICULTY OF USE

Programming the SRC-6E to use user-defined macros requires knowledge of high-level programming languages, hardware description languages, hardware component design, and synthesizability. Relatively few people possess all of these skills to use the system effectively without first receiving significant training. However, programming the system using only high-level languages of C or Fortran is possible which widens the potential user base to many more people. More research needs to be performed to determine which is most feasible for organizations using the machine.

The SRC-6E has a relatively steep learning curve. There are few examples in the documentation and little work using the system. The errors generated by the system during development are not intuitive and cannot be solved without previous experience. The SRC support staff are very helpful in solving specific code problems but are not forthcoming in the reasons or methods used to resolve them. There are no development tools in place to assist novice users in programming the system.

The development time to implement solutions on this system appears to be high, primarily due to the steep learning curve and lack of development tools. This research represents approximately six months of part-time work by a single moderately experienced person. More research must be performed to further quantify the development time and see how it improves once a group of experienced repeat users is grown. No research has yet been performed with large projects employing multiple programmers to see if the total project time can be reduced.

B. APPROPRIATENESS OF THIS ALGORITHM FOR TESTING

The implementation of the design with 64 or less range bins has been shown to lack the necessary parallelism to fully use the hardware and make it effective. Without increases in the memory size allocated for the user logic, implementation below 64 range bins on the SRC-6E is not effective in terms of development time, processing time, or cost-effectiveness.

C. APPROPRIATENESS OF THE SRC-6E FOR IMPLEMENTATION

The SRC-6E user logic is tied to a fixed 100MHz clock which would limit the range resolution if it were used to implement an actual false target radar imaging system. Range resolution is defined as [10]:

$R_R = \frac{C}{2f_{cl}}$, where *C* is the speed of light, and f_{cl} is the clock speed. With a clock speed of 100 MHz range resolution is 1.5 meters. Ref [10] reports a system would require at least a 500 MHz clock speed which results in a resolution of 0.3 meters necessary to fool modern radars. Maximum target size is defined as [10]:

$M_{SZ} = R_R \times N_{RB}$, where N_{RB} is the number of range bins.

Using 128 range bins results in a maximum target size of 192 meters. With an expected 200 range bins possible, 300 meters is achievable. With the use of two user logic areas, over 400 range bins should be achievable, resulting in 600 meters. Future versions of reconfigureable computers using more modern FPGA’s with both larger sizes and faster clock speeds could make a workable false target radar imaging system achievable with this type of technology.

D. RECOMMENDATIONS FOR FUTURE WORK

The current interface allows scalability to any number of range bins in units of 4. 256 range bins have been shown to be too large to fit within the user logic while 128 range bins fits with 68% usage. Future research could use more than one user logic chip to implement twice as many range bins.

As discussed, the SRC-6E lacks a custom code editing environment with modern features such as real-time syntax checking. Automated generation of some of the support files could also be implemented. Tools could be created that ask a few questions and then create the skeletons of the support files for the project. Changes to one file that affect another could be automatically corrected or at a minimum generate warnings.

The knowledge base of what types of applications do or do not work efficiently on this system is very small. Many more algorithms need to be tested on the system. Programming the same algorithm with both a high-level language and user macros would provide information on which produces better results for different types of algorithms. Cost and timing comparison to modern and readily available computers should continue to be made.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A

This appendix contains the VHDL code that was both tested with Aldec Active-HDL and ported to the SRC-6E. The full code is shown for the 4 bin version. Only the changed portion is shown for the other version. The 4 bin model is labeled to show the section replaced by the other version's code.

A. 4 BIN MACRO VHDL

```
------------------------------------------------------------
-- Title       : FourBin
------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity DFlipFlop is
port (CLK, LD, RESET, D: in bit;
Q: inout bit; Qnot: out bit := '1');
end DFLipFlop;

architecture Equations of DFlipFlop is
begin
process (CLK, LD, RESET)
begin
if CLK='1' and CLK'EVENT then
if  RESET='1' then
Q <= '0';
elsif LD='1' then
Q <= D;
end if;
end if;
end process;
Qnot <= not Q;
end Equations;

entity Register5 is
port (CLK,LD,RESET: in bit; D5: in bit_vector (4 downto 0);
      Q5: inout bit_vector (4 downto 0);
Q5not: out bit_vector (4 downto 0));
end Register5;

architecture Register5 of Register5 is
component DFlipFlop
port (CLK, LD, RESET, D: in bit;
Q: inout bit; Qnot: out bit);
end component;
begin
DFF0: DFlipFlop port map (CLK, LD, RESET, D5(0), Q5(0), Q5not(0));
```

```
DFF1: DFlipFlop port map (CLK, LD, RESET, D5(1), Q5(1), Q5not(1));
DFF2: DFlipFlop port map (CLK, LD, RESET, D5(2), Q5(2), Q5not(2));
DFF3: DFlipFlop port map (CLK, LD, RESET, D5(3), Q5(3), Q5not(3));
DFF4: DFlipFlop port map (CLK, LD, RESET, D5(4), Q5(4), Q5not(4));
end Register5;

entity Register8 is
port (CLK, LD, RESET: in bit; D8: in bit_vector (7 downto 0);
Q8: inout bit_vector (7 downto 0);
Q8not: out bit_vector (7 downto 0));
end Register8;

architecture Register8 of Register8 is
component DFlipFlop
port (CLK, LD, RESET, D: in bit;
Q: inout bit; Qnot: out bit);
end component;

begin
DFF0: DFlipFlop port map (CLK, LD, RESET, D8(0), Q8(0), Q8not(0));
DFF1: DFlipFlop port map (CLK, LD, RESET, D8(1), Q8(1), Q8not(1));
DFF2: DFlipFlop port map (CLK, LD, RESET, D8(2), Q8(2), Q8not(2));
DFF3: DFlipFlop port map (CLK, LD, RESET, D8(3), Q8(3), Q8not(3));
DFF4: DFlipFlop port map (CLK, LD, RESET, D8(4), Q8(4), Q8not(4));
DFF5: DFlipFlop port map (CLK, LD, RESET, D8(5), Q8(5), Q8not(5));
DFF6: DFlipFlop port map (CLK, LD, RESET, D8(6), Q8(6), Q8not(6));
DFF7: DFlipFlop port map (CLK, LD, RESET, D8(7), Q8(7), Q8not(7));
end Register8;

entity Register13 is
port (CLK, LD, RESET: in bit; D13: in bit_vector (12 downto 0);
Q13: inout bit_vector (12 downto 0);
Q13not: out bit_vector (12 downto 0));
end Register13;

architecture Register13 of Register13 is
component DFlipFlop
port (CLK, LD, RESET, D: in bit;
Q: inout bit; Qnot: out bit);
end component;

begin
DFF0: DFlipFlop port map (CLK, LD, RESET, D13(0), Q13(0), Q13not(0));
DFF1: DFlipFlop port map (CLK, LD, RESET, D13(1), Q13(1), Q13not(1));
DFF2: DFlipFlop port map (CLK, LD, RESET, D13(2), Q13(2), Q13not(2));
DFF3: DFlipFlop port map (CLK, LD, RESET, D13(3), Q13(3), Q13not(3));
DFF4: DFlipFlop port map (CLK, LD, RESET, D13(4), Q13(4), Q13not(4));
DFF5: DFlipFlop port map (CLK, LD, RESET, D13(5), Q13(5), Q13not(5));
DFF6: DFlipFlop port map (CLK, LD, RESET, D13(6), Q13(6), Q13not(6));
DFF7: DFlipFlop port map (CLK, LD, RESET, D13(7), Q13(7), Q13not(7));
DFF8: DFlipFlop port map (CLK, LD, RESET, D13(8), Q13(8), Q13not(8));
DFF9: DFlipFlop port map (CLK, LD, RESET, D13(9), Q13(9), Q13not(9));
DFF10: DFlipFlop port map (CLK, LD, RESET, D13(10), Q13(10),
Q13not(10));
DFF11: DFlipFlop port map (CLK, LD, RESET, D13(11), Q13(11),
Q13not(11));
```

```
DFF12: DFlipFlop port map (CLK, LD, RESET, D13(12), Q13(12),
Q13not(12));
end Register13;

entity Register17 is
port (CLK, LD, RESET: in bit; D17: in bit_vector (16 downto 0);
Q17: inout bit_vector (16 downto 0);
Q17not: out bit_vector (16 downto 0));
end Register17;

architecture Register17 of Register17 is
component DFlipFlop
port (CLK, LD, RESET,D: in bit;
Q: inout bit; Qnot: out bit);
end component;

begin
DFF0: DFlipFlop port map (CLK, LD, RESET, D17(0), Q17(0), Q17not(0));
DFF1: DFlipFlop port map (CLK, LD, RESET, D17(1), Q17(1), Q17not(1));
DFF2: DFlipFlop port map (CLK, LD, RESET, D17(2), Q17(2), Q17not(2));
DFF3: DFlipFlop port map (CLK, LD, RESET, D17(3), Q17(3), Q17not(3));
DFF4: DFlipFlop port map (CLK, LD, RESET, D17(4), Q17(4), Q17not(4));
DFF5: DFlipFlop port map (CLK, LD, RESET, D17(5), Q17(5), Q17not(5));
DFF6: DFlipFlop port map (CLK, LD, RESET, D17(6), Q17(6), Q17not(6));
DFF7: DFlipFlop port map (CLK, LD, RESET, D17(7), Q17(7), Q17not(7));
DFF8: DFlipFlop port map (CLK, LD, RESET, D17(8), Q17(8), Q17not(8));
DFF9: DFlipFlop port map (CLK, LD, RESET, D17(9), Q17(9), Q17not(9));
DFF10: DFlipFlop port map (CLK, LD, RESET, D17(10), Q17(10),
Q17not(10));
DFF11: DFlipFlop port map (CLK, LD, RESET, D17(11), Q17(11),
Q17not(11));
DFF12: DFlipFlop port map (CLK, LD, RESET, D17(12), Q17(12),
Q17not(12));
DFF13: DFlipFlop port map (CLK, LD, RESET, D17(13), Q17(13),
Q17not(13));
DFF14: DFlipFlop port map (CLK, LD, RESET, D17(14), Q17(14),
Q17not(14));
DFF15: DFlipFlop port map (CLK, LD, RESET, D17(15), Q17(15),
Q17not(15));
DFF16: DFlipFlop port map (CLK, LD, RESET, D17(16), Q17(16),
Q17not(16));
end Register17;

entity ROMLUT is
port (SIN, COS:out bit_vector(8 downto 1);
  FIVEBITS:in bit_vector(5 downto 1));
end ROMLUT;

architecture ROMLUT of ROMLUT is

signal ROMLUTValue : bit_vector(15 downto 0);

begin
with FIVEBITS Select
   ROMLUTValue <="0000000001111111" when "00000",--0
"0001100101111101" when "00001",--1
"0011000101110101" when "00010",--2
```

```
"0100011101101010" when "00011",--3
"0101101001011010" when "00100",--4
"0110101001000111" when "00101",--5
"0111010100110001" when "00110",--6
"0111110100011001" when "00111",--7
"0111111100000000" when "01000",--8
"0111110111100111" when "01001",--9
"0111010111001111" when "01010",--A
"0110101010111001" when "01011",--b
"0101101010100110" when "01100",--C
"0100011110010110" when "01101",--d
"0011000110001011" when "01110",--E
"0001100110000011" when "01111",--F
"0000000010000001" when "10000",--10
"1110011110000011" when "10001",--11
"1100111110001011" when "10010",--12
"1011100110010110" when "10011",--13
"1010011010100110" when "10100",--14
"1001011010111001" when "10101",--15
"1000101111001111" when "10110",--16
"1000001111100111" when "10111",--17
"1000000100000000" when "11000",--18
"1000001100011001" when "11001",--19
"1000101100110001" when "11010",--1A
"1001011001000111" when "11011",--1b
"1010011001011010" when "11100",--1C
"1011100101101010" when "11101",--1d
"1100111101110101" when "11110",--1E
"1110011101111101" when "11111",--1F
"0000000000000000" when others;--Never Occurs

   SIN <= ROMLUTValue(15 downto 8);
   COS <= ROMLUTValue(7 downto 0);
end ROMLUT;

entity FullAdder is
port (X, Y, Cin: in bit;
Cout, Sum: out bit);
end FullAdder;
architecture Equations of FullAdder is
begin
Sum <= X xor Y xor Cin;
Cout <= (X and Y) or (X and Cin) or (Y and Cin);
end Equations;

entity FullAdderOV is
port (Ci, Cout, OVin: in bit;
  Co, OVout: out bit);
end FullAdderOV;

architecture Equations of FullAdderOV is
begin
Co <= Cout;
OVout <= OVin or (Ci xor Cout);
end Equations;

entity Adder5 is
```

```
port (A, B: in bit_vector(4 downto 0); Ci: in bit;
  S: out bit_vector(4 downto 0); Co: out bit);
end Adder5;

architecture Adder5 of Adder5 is
component FullAdder
port (X, Y, Cin: in bit;
Cout, Sum: out bit);
end component;
signal C: bit_vector(4 downto 1);
begin
FA0: FullAdder port map (A(0), B(0), Ci, C(1), S(0));
FA1: FullAdder port map (A(1), B(1), C(1), C(2), S(1));
FA2: FullAdder port map (A(2), B(2), C(2), C(3), S(2));
FA3: FullAdder port map (A(3), B(3), C(3), C(4), S(3));
FA4: FullAdder port map (A(4), B(4), C(4), Co, S(4));
end Adder5;

entity CLAH4 is
port (A, B: in bit_vector(3 downto 0); Cin: in bit;
Cout: out bit);
end CLAH4;

architecture CLAH4 of CLAH4 is
signal g0, g1, g2, g3, p0, p1, p2, p3: bit;
begin
g0 <= A(0) and B(0);
p0 <= A(0) or B(0);
g1 <= A(1) and B(1);
p1 <= A(1) or B(1);
g2 <= A(2) and B(2);
p2 <= A(2) or B(2);
g3 <= A(3) and B(3);
p3 <= A(3) or B(3);
Cout <= g3 or (p3 and g2) or (p3 and p2 and g1) or (p3 and p2 and p1 and g0) or
(p3 and p2 and p1 and p0 and Cin);
end CLAH4;

entity CLAH8 is
port (A, B: in bit_vector(7 downto 0); Cin: in bit;
Cout: out bit);
end CLAH8;

architecture CLAH8 of CLAH8 is
signal g0, g1, g2, g3, g4, g5, g6, g7, p0, p1, p2, p3, p4, p5, p6, p7:
bit;
begin
g0 <= A(0) and B(0);
p0 <= A(0) or B(0);
g1 <= A(1) and B(1);
p1 <= A(1) or B(1);
g2 <= A(2) and B(2);
p2 <= A(2) or B(2);
g3 <= A(3) and B(3);
p3 <= A(3) or B(3);
g4 <= A(4) and B(4);
p4 <= A(4) or B(4);
```

```
g5 <= A(5) and B(5);
p5 <= A(5) or B(5);
g6 <= A(6) and B(6);
p6 <= A(6) or B(6);
g7 <= A(7) and B(7);
p7 <= A(7) or B(7);
Cout <= g7 or (p7 and g6) or (p7 and p6 and g5) or (p7 and p6 and p5 and g4) or
(p7 and p6 and p5 and p4 and g3) or
(p7 and p6 and p5 and p4 and p3 and g2) or
(p7 and p6 and p5 and p4 and p3 and p2 and g1) or
(p7 and p6 and p5 and p4 and p3 and p2 and p1 and g0) or
(p7 and p6 and p5 and p4 and p3 and p2 and p1 and p0 and Cin);
end CLAH8;

entity Adder16 is
port (A, B: in bit_vector(15 downto 0); Ci, OVin: in bit;
  S: out bit_vector(16 downto 0); Co: out bit);
end Adder16; --bit 16 of S is overflow

architecture Adder16 of Adder16 is
component CLAH4
port (A, B: in bit_vector(3 downto 0); Cin: in bit;
Cout: out bit);
end component;
component CLAH8
port (A, B: in bit_vector(7 downto 0); Cin: in bit;
Cout: out bit);
end component;
component FullAdder
port (X, Y, Cin: in bit;
Cout, Sum: out bit);
end component;
component FullAdderOV
port (Ci, Cout, OVin: in bit;
  Co, OVout: out bit);
end component;

signal C: bit_vector(16 downto 1);
signal dummy1, dummy2, dummy3: bit;
begin
FA0: FullAdder port map (A(0), B(0), Ci, C(1), S(0));
FA1: FullAdder port map (A(1), B(1), C(1), C(2), S(1));
FA2: FullAdder port map (A(2), B(2), C(2), C(3), S(2));
FA3: FullAdder port map (A(3), B(3), C(3), dummy1, S(3));
CLAH0: CLAH4 port map (A(3 downto 0), B(3 downto 0), Ci, C(4));
FA4: FullAdder port map (A(4), B(4), C(4), C(5), S(4));
FA5: FullAdder port map (A(5), B(5), C(5), C(6), S(5));
FA6: FullAdder port map (A(6), B(6), C(6), C(7), S(6));
FA7: FullAdder port map (A(7), B(7), C(7), dummy2, S(7));
CLAH1: CLAH8 port map (A(7 downto 0), B(7 downto 0), Ci, C(8));
FA8: FullAdder port map (A(8), B(8), C(8), C(9), S(8));
FA9: FullAdder port map (A(9), B(9), C(9), C(10), S(9));
FA10: FullAdder port map (A(10), B(10), C(10), C(11), S(10));
FA11: FullAdder port map (A(11), B(11), C(11), dummy3, S(11));
CLAH2: CLAH4 port map (A(11 downto 8), B(11 downto 8), C(8), C(12));
FA12: FullAdder port map (A(12), B(12), C(12), C(13), S(12));
FA13: FullAdder port map (A(13), B(13), C(13), C(14), S(13));
```

```
FA14: FullAdder port map (A(14), B(14), C(14), C(15), S(14));
FA15: FullAdder port map (A(15), B(15), C(15), C(16), S(15));
FAOV: FullAdderOV port map (C(15), C(16), OVin, Co, S(16));
end Adder16;

entity ControlLogic is
port (ODVin, URB, PSVin, CLK, OPER: in bit;
CLR13, CLR17: out bit := '1'; ODVout, PSVout: out bit);
end ControlLogic;

architecture ControlLogic of ControlLogic is
component DFlipFlop
port (CLK, LD, RESET, D: in bit;
Q: inout bit; Qnot: out bit);
end component;
signal RESET,D1,Q1,Q1Not,D2,Q2,Q2Not,D3,Q3,Q3Not,D4,Q4,Q4Not,
PSVD,PSVQ,PSVQNot:bit;
begin
RESET <= '0';
PSVFF: DFlipFlop port map (CLK, OPER, RESET, PSVD, PSVQ, PSVQNot);
DFF1: DFLipFlop port map(CLK, OPER, RESET, D1, Q1, Q1Not);
DFF2: DFLipFlop port map(CLK, OPER, RESET, D2, Q2, Q2Not);
DFF3: DFLipFlop port map(CLK, OPER, RESET, D3, Q3, Q3Not);
DFF4: DFLipFlop port map(CLK, OPER, RESET, D4, Q4, Q4Not);
PSVD <= PSVin;
D1 <= URB and PSVQ;
D2 <= Q1;
D3 <= ODVin or Q2;
D4 <= Q3;
CLR13 <= Q2Not;
CLR17 <= Q3Not;
PSVout <= PSVQ;
ODVout <= Q4;
end ControlLogic;

entity GainShifter is
port (Control:in bit_vector(4 downto 1);
Data: in bit_vector(8 downto 1);
Output: out bit_vector(13 downto 1));
end GainShifter;

architecture GainShifter of GainShifter is
begin
process (Control,Data)
begin
Output(13 downto 1) <= "0000000000000";
case Control is
when "0000" => Output(3 downto 1) <= Data(8 downto 6);
if Data(8)='1' then --need to preserve the sign bit
Output(13 downto 4) <= "1111111111";
end if;
when "0001" => Output(4 downto 1) <= Data(8 downto 5);
if Data(8)='1' then --need to preserve the sign bit
Output(13 downto 5) <= "111111111";
end if;
when "0010" => Output(5 downto 1) <= Data(8 downto 4);
if Data(8)='1' then --need to preserve the sign bit
```

```
Output(13 downto 6) <= "11111111";
end if;
when "0011" => Output(6 downto 1) <= Data(8 downto 3);
if Data(8)='1' then --need to preserve the sign bit
Output(13 downto 7) <= "1111111";
end if;
when "0100" => Output(6 downto 1) <= Data(8 downto 3);
if Data(8)='1' then --need to preserve the sign bit
Output(13 downto 7) <= "1111111";
end if;
when "0101" => Output(7 downto 1) <= Data(8 downto 2);
if Data(8)='1' then --need to preserve the sign bit
Output(13 downto 8) <= "111111";
end if;
when "0110" => Output(8 downto 1) <= Data(8 downto 1);
if Data(8)='1' then --need to preserve the sign bit
Output(13 downto 9) <= "11111";
end if;
when "0111" => Output(9 downto 2) <= Data(8 downto 1);
if Data(8)='1' then --need to preserve the sign bit
Output(13 downto 10) <= "1111";
end if;
when "1000" => Output(7 downto 1) <= Data(8 downto 2);
if Data(8)='1' then --need to preserve the sign bit
Output(13 downto 8) <= "111111";
end if;
when "1001" => Output(8 downto 1) <= Data(8 downto 1);
if Data(8)='1' then --need to preserve the sign bit
Output(13 downto 9) <= "11111";
end if;
when "1010" => Output(9 downto 2) <= Data(8 downto 1);
if Data(8)='1' then --need to preserve the sign bit
Output(13 downto 10) <= "1111";
end if;
when "1011" => Output(10 downto 3) <= Data(8 downto 1);
if Data(8)='1' then --need to preserve the sign bit
Output(13 downto 11) <= "111";
end if;
when "1100" => Output(10 downto 3) <=Data(8 downto 1);
if Data(8)='1' then --need to preserve the sign bit
Output(13 downto 11) <= "111";
end if;
when "1101" => Output(11 downto 4) <=Data(8 downto 1);
if Data(8)='1' then --need to preserve the sign bit
Output(13 downto 12) <= "11";
end if;
when "1110" => Output(12 downto 5) <=Data(8 downto 1);
if Data(8)='1' then --need to preserve the sign bit
Output(13) <= '1';
end if;
when "1111" => Output(13 downto 6) <=Data(8 downto 1);
when others => -- summon blue screen of death
end case;
end process;
end GainShifter;
entity OneBin is
port (PhaseSamp, PhaseInc: in bit_vector (5 downto 1);
Gain: in bit_vector (4 downto 1);
URB, UNP, PRB, ODVin, PSVin: in bit;
OtherBinDataSIN, OtherBinDataCOS: in bit_vector (17 downto 1);
Q, I: out bit_vector (17 downto 1);
ODVout, PSVout: out bit;
DRFM: out bit_vector (5 downto 1);
CLK: in bit);
end OneBin;

architecture OneBin of OneBin is
component Register5 is
port (CLK, LD, RESET: in bit; D5: in bit_vector (4 downto 0);
Q5: inout bit_vector (4 downto 0); Q5not: out bit_vector (4 downto 0));
end component;
component Register8 is
port (CLK, LD, RESET: in bit; D8: in bit_vector (7 downto 0);
Q8: inout bit_vector (7 downto 0); Q8not: out bit_vector (7 downto 0));
end component;
component Register13 is
port (CLK, LD, RESET: in bit; D13: in bit_vector (12 downto 0);
Q13: inout bit_vector(12 downto 0);Q13not:out bit_vector(12 downto 0));
end component;
component Register17 is
port (CLK, LD, RESET: in bit; D17: in bit_vector (16 downto 0);
Q17: inout bit_vector(16 downto 0);Q17not:out bit_vector(16 downto 0));
end component;
component Adder5 is
port (A, B: in bit_vector(4 downto 0); Ci: in bit;
  S: out bit_vector(4 downto 0); Co: out bit);
end component;
component Adder16 is
port (A, B: in bit_vector(15 downto 0); Ci, OVin: in bit;
  S: out bit_vector(16 downto 0); Co: out bit);
end component;
component ROMLUT is
port (SIN, COS:out bit_vector(1 to 8);
  FIVEBITS:in bit_vector(1 to 5));
end component;
component GainShifter is
port (Control:in bit_vector(4 downto 1);Data:in bit_vector(8 downto 1);
Output: out bit_vector(13 downto 1));
end component;
component ControlLogic is
port (ODVin, URB, PSVin, CLK, OPER: in bit;
  CLR13, CLR17, ODVout, PSVout: out bit);
end component;

signal
QOutReg1,QNotOutReg1,QOutReg2,QNotOutReg2,QOutReg3,QNotOutReg3,QOutReg4
,QNotOutReg4,
QOutReg5,QNotOutReg5,QOutReg6,QNotOutReg6,OutAdd1: bit_vector (5 downto
1);
signal QOutReg7,QNotOutReg7,QOutReg8,QNotOutReg8,LUTSIN, LUTCOS:
bit_vector (8 downto 1);
signal QOutReg9,QNotOutReg9,QOutReg10,QNotOutReg10, OutShiftSIN,
OutShiftCOS: bit_vector (13 downto 1);
```

```
signal
QOutReg11,QNotOutReg11,QOutReg12,QNotOutReg12,QOutReg13,QNotOutReg13,QO
utReg14,QNotOutReg14,
OutAdd2, OutAdd3: bit_vector (17 downto 1);
signal InputAdder2, InputAdder3: bit_vector (16 downto 1);
signal OPER, LD, CLR5, CLR8, CLR13, CLR17, Ci, Co1, Co2, Co3,
Reset_Inact: bit;
signal InReg5: bit_vector (5 downto 1);
begin
OPER <='1';
CLR5 <= '0';
CLR8 <= '0';
Ci <= '0';
LD <= '1';
Reset_Inact <= '0';
InReg5(4 downto 1) <= Gain (4 downto 1);
InReg5(5) <= URB;
Reg1: Register5 port map(CLK, PRB, CLR5, PhaseInc(5 downto 1),
QOutReg1(5 downto 1), QNotOutReg1(5 downto 1));
Reg2: Register5 port map(CLK, UNP, CLR5, QOutReg1(5 downto 1),
QOutReg2(5 downto 1), QNotOutReg2(5 downto 1));
Reg3: Register5 port map(CLK, LD, CLR5, PhaseSamp(5 downto 1),
QOutReg3(5 downto 1), QNotOutReg3(5 downto 1));
Add1: Adder5 port map (QOutReg2,QOutReg3, Ci, OutAdd1(5 downto 1),
Co1);
Reg4: Register5 port map(CLK, LD, CLR5, OutAdd1(5 downto 1),
QOutReg4(5 downto 1), QNotOutReg4(5 downto 1));
LUT: ROMLUT port map (LUTSIN(8 downto 1),LUTCOS(8 downto 1),
QOutReg4(5 downto 1));
Reg5: Register5 port map(CLK, PRB, CLR5, InReg5(5 downto 1),
QOutReg5(5 downto 1), QNotOutReg5(5 downto 1));
Reg6: Register5 port map(CLK, UNP, CLR5, QOutReg5(5 downto 1),
QOutReg6(5 downto 1), QNotOutReg6(5 downto 1));
Reg7: Register8 port map(CLK, LD, CLR8, LUTSIN(8 downto 1),
QOutReg7(8 downto 1), QNotOutReg7(8 downto 1));
Reg8: Register8 port map(CLK, LD, CLR8, LUTCOS(8 downto 1),
QOutReg8(8 downto 1), QNotOutReg8(8 downto 1));
Shift1: GainShifter port map (QOutReg6(4 downto 1),QOutReg7(8 downto 1),OutShiftSIN(13
Shift2: GainShifter port map (QOutReg6(4 downto 1),QOutReg8(8 downto 1),OutShiftCOS(13
Reg9: Register13 port map(CLK, LD, CLR13, OutShiftSIN(13 downto 1),QOutReg9(13 downto 1
Reg10: Register13 port map(CLK, LD, CLR13, OutShiftCOS(13 downto 1), QOutReg10(13 downt
Reg11: Register17 port map(CLK, LD, Reset_Inact, OtherBinDataSIN(17 downto 1), QOutReg1
Reg12: Register17 port map(CLK, LD, Reset_Inact, OtherBinDataCOS(17 downto 1), QOutReg1
Add2: Adder16 port map (InputAdder2, QOutReg11(16 downto 1), Ci, QOutReg11(17), OutAdd2
Add3: Adder16 port map (InputAdder3, QOutReg12(16 downto 1), Ci, QOutReg12(17), OutAdd3
Reg13: Register17 port map(CLK, LD, CLR17, OutAdd2(17 downto 1), QOutReg13(17 downto 1)
Reg14: Register17 port map(CLK, LD, CLR17, OutAdd3(17 downto 1), QOutReg14(17 downto 1)
Control: ControlLogic port map (ODVin, QOutReg6(5), PSVin, CLK,OPER, CLR13, CLR17, ODVo
InputAdder2(13 downto 1) <= QOutReg9(13 downto 1);
InputAdder2(14) <= QOutReg9(13);
InputAdder2(15) <= QOutReg9(13);
InputAdder2(16) <= QOutReg9(13);
InputAdder3(13 downto 1) <= QOutReg10(13 downto 1);
InputAdder3(14) <= QOutReg10(13);
InputAdder3(15) <= QOutReg10(13);
InputAdder3(16) <= QOutReg10(13);
DRFM(5 downto 1) <= QOutReg3(5 downto 1);
```

```
Q <= QOutReg13;
I <= QOutReg14;
end OneBin;

--**** BEGIN REPLACE HERE *****

entity FourBin is
port (Input: in bit_vector (63 downto 0);
Output:out bit_vector (63 downto 0); CLK: in bit);
end FourBin;

architecture FourBin of FourBin is

component OneBin is
port (PhaseSamp, PhaseInc: in bit_vector (5 downto 1);
Gain: in bit_vector (4 downto 1);
URB, UNP, PRB, ODVin, PSVin: in bit;
OtherBinDataSIN,OtherBinDataCOS:in bit_vector (17 downto 1);
Q, I: out bit_vector (17 downto 1);
ODVout, PSVout: out bit;
DRFM: out bit_vector (5 downto 1);
CLK: in bit);
end component;

signal Q, I, Q1,I1,Q2,I2,Q3,I3: bit_vector (17 downto 1)
:="00000000000000000";
signal DRFM0,DRFM1,DRFM2,DRFM3: bit_vector (5 downto 1):="00000";
signal PSVout0,PSVout1,PSVout2,PSVout3: bit:='0';
signal ODVout0,ODVout1,ODVout2,ODVout3: bit:='0';
signal Gain0,Gain1,Gain2,Gain3: bit_vector (4 downto 1):="0000";
signal PhaseInc0,PhaseInc1,PhaseInc2,PhaseInc3: bit_vector (5 downto 1)
:="00000";
signal URB0,URB1,URB2,URB3: bit :='0';
signal PhaseSamp: bit_vector (5 downto 1):= "00000";
signal PRB, UNP, PSVin, ODVin: bit :='0';

begin -- BIN0 is the primary output
--Input: formats, cycle begins when PRB=1, 1 format per clock
-- ten bits of each bin: 10 |  9-5   |4-1 |
--
-- 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
--unused | bin3 | bin2 | bin1 | bin0 |sample |PSVin |UNP |PRB
-- 63-8                              | 7-3   | 2    | 1  | 0
--unused                             |sample |PSVin |UNP |PRB
-- The first sample can be entered on clock 2 if PSVin and UNP are
taken high

PRB<=Input(0);
UNP<=Input(1);
PSVin<=Input(2);
PhaseSamp<=Input(7 downto 3);
-- Bin 0 data
URB0<=Input(17);
PhaseInc0<=Input(16 downto 12);
Gain0<=Input(11 downto 8);
-- Bin 1 Data
URB1<=Input(27);
```

```
PhaseInc1<=Input(26 downto 22);
Gain1<=Input(21 downto 18);
-- Bin 2 data
URB2<=Input(37);
PhaseInc2<=Input(36 downto 32);
Gain2<=Input(31 downto 28);
-- Bins 3 data
URB3<=Input(47);
PhaseInc3<=Input(46 downto 42);
Gain3<=Input(41 downto 38);

BIN0: OneBin port map (DRFM1, PhaseInc0, Gain0, URB0, UNP, PRB,
ODVout1, PSVout1, Q1, I1, Q, I, ODVout0, PSVout0, DRFM0, CLK);
BIN1: OneBin port map (DRFM2, PhaseInc1, Gain1, URB1, UNP, PRB,
ODVout2, PSVout2, Q2, I2, Q1, I1, ODVout1, PSVout1, DRFM1, CLK);
BIN2: OneBin port map (DRFM3, PhaseInc2, Gain2, URB2, UNP, PRB,
ODVout3, PSVout3, Q3, I3, Q2, I2, ODVout2, PSVout2, DRFM2, CLK);
BIN3: OneBin port map (PhaseSamp, PhaseInc3, Gain3, URB3, UNP, PRB,
ODVin, PSVin, "00000000000000000", "00000000000000000", Q3, I3,
ODVout3, PSVout3, DRFM3, CLK);

-- Output Format:
-- 63-41 |   40   |   39   | 38-22 | 21-5 | 4-0
--unused | PSVout | ODVout |   Q   |   I  | DRFM

Output(40)<=PSVout0;
Output(39)<=ODVout0;
Output(38 downto 22)<=Q;
Output(21 downto 5)<=I;
Output(4 downto 0)<=DRFM0;
Output(63 downto 41)<="00000000000000000000000";

end FourBin;

--**** END REPLACE HERE *****
```

B. 8 BIN MACRO VHDL

```
entity EightBin is
port (Input: in bit_vector (63 downto 0);
Output:out bit_vector (63 downto 0); CLK: in bit);
end EightBin;

architecture EightBin of EightBin is

component OneBin is
port (PhaseSamp, PhaseInc: in bit_vector (5 downto 1);
Gain: in bit_vector (4 downto 1);
URB, UNP, PRB, ODVin, PSVin: in bit;
OtherBinDataSIN,OtherBinDataCOS:in bit_vector (17 downto 1);
Q, I: out bit_vector (17 downto 1);
ODVout, PSVout: out bit;
DRFM: out bit_vector (5 downto 1);
CLK: in bit);
end component;
component DFlipFlop
port (CLK, LD, RESET, D: in bit;
Q: inout bit; Qnot: out bit);
end component;
signal Q, I, Q1,I1,Q2,I2,Q3,I3,Q4,I4,Q5,I5,Q6,I6,Q7,I7: bit_vector (17
downto 1):="00000000000000000";
signal DRFM0,DRFM1,DRFM2,DRFM3,DRFM4,DRFM5,DRFM6,DRFM7: bit_vector (5
downto 1):="00000";
signal PSVout0,PSVout1,PSVout2,PSVout3,PSVout4,PSVout5,PSVout6,PSVout7:
bit:='0';
signal ODVout0,ODVout1,ODVout2,ODVout3,ODVout4,ODVout5,ODVout6,ODVout7:
bit:='0';
signal Gain0,Gain1,Gain2,Gain3,Gain4,Gain5,Gain6,Gain7: bit_vector (4
downto 1):="0000";
signal PhaseInc0,PhaseInc1,PhaseInc2,PhaseInc3,PhaseInc4,
PhaseInc5,PhaseInc6,PhaseInc7 : bit_vector (5 downto 1):="00000";
signal URB0,URB1,URB2,URB3,URB4,URB5,URB6,URB7: bit :='0';
signal PhaseSamp: bit_vector (5 downto 1):= "00000";
signal UNP, PSVin, ODVin: bit :='0';
signal LD, RESET, PRB, PRB1, PRB1not: bit;

begin -- BIN0 is the primary output
--Input: formats, cycle begins when PRB=1, 1 format per clock
-- ten bits of each bin: 10 |  9-5   |4-1 |
--
-- 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
--unused | bin3 | bin2 | bin1 | bin0 |sample |PSVin |UNP |PRB
-- 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
--unused | bin7 | bin6 | bin5 | bin4 |sample |PSVin |UNP |PRB
-- 63-8                             | 7-3   | 2    | 1  | 0
--unused                            |sample |PSVin |UNP |PRB
-- The first sample can be entered on clock 3 if PSVin and UNP are
taken high
LD<='1';
RESET<='0';
PRB<=Input(0);
UNP<=Input(1);
PSVin<=Input(2);
PhaseSamp<=Input(7 downto 3);
-- Bins 0,4 data
URB0<=Input(17);
PhaseInc0<=Input(16 downto 12);
Gain0<=Input(11 downto 8);
URB4<=Input(17);
PhaseInc4<=Input(16 downto 12);
Gain4<=Input(11 downto 8);
-- Bins 1,5 data
URB1<=Input(27);
PhaseInc1<=Input(26 downto 22);
Gain1<=Input(21 downto 18);
URB5<=Input(27);
PhaseInc5<=Input(26 downto 22);
Gain5<=Input(21 downto 18);
-- Bins 2,6 data
URB2<=Input(37);
PhaseInc2<=Input(36 downto 32);
Gain2<=Input(31 downto 28);
URB6<=Input(37);
PhaseInc6<=Input(36 downto 32);
```

```
Gain6<=Input(31 downto 28);
-- Bins 3,7 data
URB3<=Input(47);
PhaseInc3<=Input(46 downto 42);
Gain3<=Input(41 downto 38);
URB7<=Input(47);
PhaseInc7<=Input(46 downto 42);
Gain7<=Input(41 downto 38);

DFF: DFlipFlop port map (CLK, LD, RESET, PRB, PRB1, PRB1not);
BIN0: OneBin port map (DRFM1, PhaseInc0, Gain0, URB0, UNP, PRB,
ODVout1, PSVout1, Q1, I1, Q, I, ODVout0, PSVout0, DRFM0, CLK);
BIN1: OneBin port map (DRFM2, PhaseInc1, Gain1, URB1, UNP, PRB,
ODVout2, PSVout2, Q2, I2, Q1, I1, ODVout1, PSVout1, DRFM1, CLK);
BIN2: OneBin port map (DRFM3, PhaseInc2, Gain2, URB2, UNP, PRB,
ODVout3, PSVout3, Q3, I3, Q2, I2, ODVout2, PSVout2, DRFM2, CLK);
BIN3: OneBin port map (DRFM4, PhaseInc3, Gain3, URB3, UNP, PRB,
ODVout4, PSVout4, Q4, I4, Q3, I3, ODVout3, PSVout3, DRFM3, CLK);
BIN4: OneBin port map (DRFM5, PhaseInc4, Gain4, URB4, UNP, PRB1,
ODVout5, PSVout5, Q5, I5, Q4, I4, ODVout4, PSVout4, DRFM4, CLK);
BIN5: OneBin port map (DRFM6, PhaseInc5, Gain5, URB5, UNP, PRB1,
ODVout6, PSVout6, Q6, I6, Q5, I5, ODVout5, PSVout5, DRFM5, CLK);
BIN6: OneBin port map (DRFM7, PhaseInc6, Gain6, URB6, UNP, PRB1,
ODVout7, PSVout7, Q7, I7, Q6, I6, ODVout6, PSVout6, DRFM6, CLK);
BIN7: OneBin port map (PhaseSamp, PhaseInc7, Gain7, URB7, UNP, PRB1,
ODVin, PSVin, "00000000000000000", "00000000000000000", Q7, I7,
ODVout7, PSVout7, DRFM7, CLK);
-- Output Format:
-- 63-41 |   40   |   39   | 38-22 | 21-5 | 4-0
--unused | PSVout | ODVout |   Q   |   I  | DRFM

Output(40)<=PSVout0;
Output(39)<=ODVout0;
Output(38 downto 22)<=Q;
Output(21 downto 5)<=I;
Output(4 downto 0)<=DRFM0;
Output(63 downto 41)<="00000000000000000000000";

end EightBin;
```

C. 16 BIN MACRO VHDL

```
entity SixteenBin is
port (Input: in bit_vector (63 downto 0);
Output:out bit_vector (63 downto 0); CLK: in bit);
end SixteenBin;

architecture SixteenBin of SixteenBin is

component OneBin is
port (PhaseSamp, PhaseInc: in bit_vector (5 downto 1);
Gain: in bit_vector (4 downto 1);
URB, UNP, PRB, ODVin, PSVin: in bit;
OtherBinDataSIN,OtherBinDataCOS:in bit_vector (17 downto 1);
Q, I: out bit_vector (17 downto 1);
ODVout, PSVout: out bit;
DRFM: out bit_vector (5 downto 1);
CLK: in bit);
end component;
component DFlipFlop
port (CLK, LD, RESET, D: in bit;
Q: inout bit; Qnot: out bit);
end component;

signal Q, I, Q1,I1,Q2,I2,Q3,I3,Q4,I4,Q5,I5,Q6,I6,Q7,I7,Q8,I8,Q9,I9,
Q10,I10,Q11,I11,Q12,I12,Q13,I13,Q14,I14, Q15,I15: bit_vector (17 downto
1):="00000000000000000";
signal DRFM0,DRFM1,DRFM2,DRFM3,DRFM4,DRFM5,DRFM6,DRFM7,DRFM8,DRFM9,
DRFM10,DRFM11,DRFM12,DRFM13,DRFM14,DRFM15: bit_vector (5 downto
1):="00000";
signal PSVout0,PSVout1,PSVout2,PSVout3,PSVout4,PSVout5,PSVout6,
PSVout7,PSVout8,PSVout9,PSVout10,PSVout11,PSVout12,PSVout13,PSVout14,PS
Vout15: bit:='0';
signal ODVout0,ODVout1,ODVout2,ODVout3,ODVout4,ODVout5,ODVout6,
ODVout7,ODVout8,ODVout9,ODVout10,ODVout11,ODVout12,ODVout13,ODVout14,OD
Vout15: bit:='0';
signal Gain0,Gain1,Gain2,Gain3,Gain4,Gain5,Gain6,Gain7,Gain8,Gain9,
Gain10,Gain11,Gain12,Gain13,Gain14,Gain15: bit_vector (4 downto
1):="0000";
Signal
PhaseInc0,PhaseInc1,PhaseInc2,PhaseInc3,PhaseInc4,PhaseInc5,PhaseInc6,
PhaseInc7 : bit_vector (5 downto 1):="00000";
signal PhaseInc8,PhaseInc9,PhaseInc10,PhaseInc11,PhaseInc12,PhaseInc13,
PhaseInc14,PhaseInc15 : bit_vector (5 downto 1):="00000";
signal URB0,URB1,URB2,URB3,URB4,URB5,URB6,URB7,URB8,URB9,URB10,URB11,
URB12,URB13,URB14,URB15: bit :='0';
signal PhaseSamp: bit_vector (5 downto 1):= "00000";
signal UNP, PSVin, ODVin: bit :='0';
signal LD, RESET, PRB, PRB1, PRB2, PRB3, PRB1not,PRB2not,PRB3not: bit;

begin -- BIN0 is the primary output
--Input: formats, cycle begins when PRB=1, 1 format per clock
-- ten bits of each bin: 10 |  9-5   |4-1 |
--
-- 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
--unused | bin3 | bin2 | bin1 | bin0 |sample |PSVin |UNP |PRB
-- 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
--unused | bin7 | bin6 | bin5 | bin4 |sample |PSVin |UNP |PRB
-- 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
--unused | bin11| bin10| bin9 | bin8 |sample |PSVin |UNP |PRB
-- 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
--unused | bin15| bin14| bin13| bin12|sample |PSVin |UNP |PRB
-- 63-8                             | 7-3   | 2    | 1  | 0
--unused                            |sample |PSVin |UNP |PRB
-- The first sample can be entered on clock 5 if PSVin and UNP are
taken high
LD<='1';
RESET<='0';
PRB<=Input(0);
UNP<=Input(1);
PSVin<=Input(2);
PhaseSamp<=Input(7 downto 3);
-- Bins 0,4,8,12 data
URB0<=Input(17);
```

```
PhaseInc0<=Input(16 downto 12);
Gain0<=Input(11 downto 8);
URB4<=Input(17);
PhaseInc4<=Input(16 downto 12);
Gain4<=Input(11 downto 8);
URB8<=Input(17);
PhaseInc8<=Input(16 downto 12);
Gain8<=Input(11 downto 8);
URB12<=Input(17);
PhaseInc12<=Input(16 downto 12);
Gain12<=Input(11 downto 8);
-- Bins 1,5,9,13 data
URB1<=Input(27);
PhaseInc1<=Input(26 downto 22);
Gain1<=Input(21 downto 18);
URB5<=Input(27);
PhaseInc5<=Input(26 downto 22);
Gain5<=Input(21 downto 18);
URB9<=Input(27);
PhaseInc9<=Input(26 downto 22);
Gain9<=Input(21 downto 18);
URB13<=Input(27);
PhaseInc13<=Input(26 downto 22);
Gain13<=Input(21 downto 18);
-- Bins 2,6,10,14 data
URB2<=Input(37);
PhaseInc2<=Input(36 downto 32);
Gain2<=Input(31 downto 28);
URB6<=Input(37);
PhaseInc6<=Input(36 downto 32);
Gain6<=Input(31 downto 28);
URB10<=Input(37);
PhaseInc10<=Input(36 downto 32);
Gain10<=Input(31 downto 28);
URB14<=Input(37);
PhaseInc14<=Input(36 downto 32);
Gain14<=Input(31 downto 28);
-- Bins 3,7,11,15 data
URB3<=Input(47);
PhaseInc3<=Input(46 downto 42);
Gain3<=Input(41 downto 38);
URB7<=Input(47);
PhaseInc7<=Input(46 downto 42);
Gain7<=Input(41 downto 38);
URB11<=Input(47);
PhaseInc11<=Input(46 downto 42);
Gain11<=Input(41 downto 38);
URB15<=Input(47);
PhaseInc15<=Input(46 downto 42);
Gain15<=Input(41 downto 38);

DFF0: DFlipFlop port map (CLK, LD, RESET, PRB, PRB1, PRB1not);
DFF1: DFlipFlop port map (CLK, LD, RESET, PRB1, PRB2, PRB2not);
DFF2: DFlipFlop port map (CLK, LD, RESET, PRB2, PRB3, PRB3not);
BIN0: OneBin port map (DRFM1, PhaseInc0, Gain0, URB0, UNP, PRB,
ODVout1, PSVout1, Q1, I1, Q, I, ODVout0, PSVout0, DRFM0, CLK);
```

```
BIN1: OneBin port map (DRFM2, PhaseInc1, Gain1, URB1, UNP, PRB,
ODVout2, PSVout2, Q2, I2, Q1, I1, ODVout1, PSVout1, DRFM1, CLK);
BIN2: OneBin port map (DRFM3, PhaseInc2, Gain2, URB2, UNP, PRB,
ODVout3, PSVout3, Q3, I3, Q2, I2, ODVout2, PSVout2, DRFM2, CLK);
BIN3: OneBin port map (DRFM4, PhaseInc3, Gain3, URB3, UNP, PRB,
ODVout4, PSVout4, Q4, I4, Q3, I3, ODVout3, PSVout3, DRFM3, CLK);
BIN4: OneBin port map (DRFM5, PhaseInc4, Gain4, URB4, UNP, PRB1,
ODVout5, PSVout5, Q5, I5, Q4, I4, ODVout4, PSVout4, DRFM4, CLK);
BIN5: OneBin port map (DRFM6, PhaseInc5, Gain5, URB5, UNP, PRB1,
ODVout6, PSVout6, Q6, I6, Q5, I5, ODVout5, PSVout5, DRFM5, CLK);
BIN6: OneBin port map (DRFM7, PhaseInc6, Gain6, URB6, UNP, PRB1,
ODVout7, PSVout7, Q7, I7, Q6, I6, ODVout6, PSVout6, DRFM6, CLK);
BIN7: OneBin port map (DRFM8, PhaseInc7, Gain7, URB7, UNP, PRB1,
ODVout8, PSVout8, Q8, I8, Q7, I7, ODVout7, PSVout7, DRFM7, CLK);
BIN8: OneBin port map (DRFM9, PhaseInc8, Gain8, URB8, UNP, PRB2,
ODVout9, PSVout9, Q9, I9, Q8, I8, ODVout8, PSVout8, DRFM8, CLK);
BIN9: OneBin port map (DRFM10, PhaseInc9, Gain9, URB9, UNP, PRB2,
ODVout10, PSVout10, Q10, I10, Q9, I9, ODVout9, PSVout9, DRFM9, CLK);
BIN10: OneBin port map (DRFM11, PhaseInc10, Gain10, URB10, UNP, PRB2,
ODVout11, PSVout11, Q11, I11, Q10, I10, ODVout10, PSVout10, DRFM10,
CLK);
BIN11: OneBin port map (DRFM12, PhaseInc11, Gain11, URB11, UNP, PRB2,
ODVout12, PSVout12, Q12, I12, Q11, I11, ODVout11, PSVout11, DRFM11,
CLK);
BIN12: OneBin port map (DRFM13, PhaseInc12, Gain12, URB12, UNP, PRB3,
ODVout13, PSVout13, Q13, I13, Q12, I12, ODVout12, PSVout12, DRFM12,
CLK);
BIN13: OneBin port map (DRFM14, PhaseInc13, Gain13, URB13, UNP, PRB3,
ODVout14, PSVout14, Q14, I14, Q13, I13, ODVout13, PSVout13, DRFM13,
CLK);
BIN14: OneBin port map (DRFM15, PhaseInc14, Gain14, URB14, UNP, PRB3,
ODVout15, PSVout15, Q15, I15, Q14, I14, ODVout14, PSVout14, DRFM14,
CLK);
BIN15: OneBin port map (PhaseSamp, PhaseInc15, Gain15, URB15, UNP,
PRB3, ODVin, PSVin, "00000000000000000", "00000000000000000", Q15, I15,
ODVout15, PSVout15, DRFM15, CLK);

-- Output Format:
-- 63-41 |   40   |   39   | 38-22 | 21-5 | 4-0
--unused | PSVout | ODVout |   Q   |   I  | DRFM

Output(40)<=PSVout0;
Output(39)<=ODVout0;
Output(38 downto 22)<=Q;
Output(21 downto 5)<=I;
Output(4 downto 0)<=DRFM0;
Output(63 downto 41)<="00000000000000000000000";

end SixteenBin;
```

D. 64 BIN MACRO VHDL

```
entity SixtyFourBin is
port (Input: in bit_vector (63 downto 0);
Output:out bit_vector (63 downto 0); CLK: in bit);
end SixtyFourBin;
```

```
architecture SixtyFourBin of SixtyFourBin is

component OneBin is
port (PhaseSamp, PhaseInc: in bit_vector (5 downto 1);
Gain: in bit_vector (4 downto 1);
URB, UNP, PRB, ODVin, PSVin: in bit;
OtherBinDataSIN,OtherBinDataCOS:in bit_vector (17 downto 1);
Q, I: out bit_vector (17 downto 1);
ODVout, PSVout: out bit;
DRFM: out bit_vector (5 downto 1);
CLK: in bit);
end component;
component DFlipFlop
port (CLK, LD, RESET, D: in bit;
Q: inout bit; Qnot: out bit);
end component;

signal Q, I, Q1,I1,Q2,I2,Q3,I3,Q4,I4,Q5,I5,Q6,I6,Q7,I7,Q8,I8,Q9,I9,
Q10,I10,Q11,I11,Q12,I12,Q13,I13,Q14,I14,Q15,I15: bit_vector (17 downto
1):="00000000000000000";
signal Q16, I16, Q17,I17,Q18,I18,Q19,I19,Q20,I20,Q21,I21,Q22,I22,Q23,
I23,Q24,I24,Q25,I25,
Q26,I26,Q27,I27,Q28,I28,Q29,I29,Q30,I30,Q31,I31:bit_vector (17 downto
1):="00000000000000000";
signal Q32, I32, Q33,I33,Q34,I34,Q35,I35,Q36,I36,Q37,I37,Q38,I38,
Q39,I39,Q40,I40,Q41,I41,
Q42,I42,Q43,I43,Q44,I44,Q45,I45,Q46,I46,Q47,I47: bit_vector (17 downto
1):="00000000000000000";
signal Q48, I48, Q49,I49,Q50,I50,Q51,I51,Q52,I52,Q53,I53,Q54,I54,Q55,
I55,Q56,I56,Q57,I57,
Q58,I58,Q59,I59,Q60,I60,Q61,I61,Q62,I62,Q63,I63: bit_vector (17 downto
1):="00000000000000000";
signal DRFM0,DRFM1,DRFM2,DRFM3,DRFM4,DRFM5,DRFM6,DRFM7,DRFM8,DRFM9,
DRFM10,DRFM11,DRFM12,DRFM13,DRFM14,DRFM15: bit_vector (5 downto
1):="00000";
signal
DRFM16,DRFM17,DRFM18,DRFM19,DRFM20,DRFM21,DRFM22,DRFM23,DRFM24,DRFM25,
DRFM26,DRFM27,DRFM28,DRFM29,DRFM30,DRFM31: bit_vector (5 downto
1):="00000";
signal
DRFM32,DRFM33,DRFM34,DRFM35,DRFM36,DRFM37,DRFM38,DRFM39,DRFM40,DRFM41,
DRFM42,DRFM43,DRFM44,DRFM45,DRFM46,DRFM47: bit_vector (5 downto
1):="00000";
signal
DRFM48,DRFM49,DRFM50,DRFM51,DRFM52,DRFM53,DRFM54,DRFM55,DRFM56,DRFM57,
DRFM58,DRFM59,DRFM60,DRFM61,DRFM62,DRFM63: bit_vector (5 downto
1):="00000";
signal PSVout0,PSVout1,PSVout2,PSVout3,PSVout4,PSVout5,PSVout6,
PSVout7,PSVout8,PSVout9,PSVout10,PSVout11,PSVout12,PSVout13,PSVout14,
PSVout15: bit:='0';
signal PSVout16,PSVout17,PSVout18,PSVout19,PSVout20,PSVout21,PSVout22,
PSVout23,PSVout24,PSVout25,PSVout26,PSVout27,PSVout28,PSVout29,PSVout30
,PSVout31: bit:='0';
signal PSVout32,PSVout33,PSVout34,PSVout35,PSVout36,PSVout37,PSVout38,
PSVout39,PSVout40,PSVout41,PSVout42,PSVout43,PSVout44,PSVout45,PSVout46
,PSVout47: bit:='0';
signal PSVout48,PSVout49,PSVout50,PSVout51,PSVout52,PSVout53,PSVout54,
PSVout55,PSVout56,PSVout57,PSVout58,PSVout59,PSVout60,PSVout61,PSVout62
,PSVout63: bit:='0';
signal ODVout0,ODVout1,ODVout2,ODVout3,ODVout4,ODVout5,ODVout6,
ODVout7,ODVout8,ODVout9,ODVout10,ODVout11,ODVout12,ODVout13,ODVout14,OD
Vout15: bit:='0';
signal ODVout16,ODVout17,ODVout18,ODVout19,ODVout20,ODVout21,ODVout22,
ODVout23,ODVout24,ODVout25,ODVout26,ODVout27,ODVout28,ODVout29,ODVout30
,ODVout31: bit:='0';
signal ODVout32,ODVout33,ODVout34,ODVout35,ODVout36,ODVout37,ODVout38,
ODVout39,ODVout40,ODVout41,ODVout42,ODVout43,ODVout44,ODVout45,ODVout46
,ODVout47: bit:='0';
signal ODVout48,ODVout49,ODVout50,ODVout51,ODVout52,ODVout53,ODVout54,
ODVout55,ODVout56,ODVout57,ODVout58,ODVout59,ODVout60,ODVout61,ODVout62
,ODVout63: bit:='0';
signal Gain0,Gain1,Gain2,Gain3,Gain4,Gain5,Gain6,Gain7,Gain8,Gain9,
Gain10,Gain11,Gain12,Gain13,Gain14,Gain15: bit_vector (4 downto
1):="0000";
signal
Gain16,Gain17,Gain18,Gain19,Gain20,Gain21,Gain22,Gain23,Gain24,Gain25,
Gain26,Gain27,Gain28,Gain29,Gain30,Gain31: bit_vector (4 downto
1):="0000";
signal
Gain32,Gain33,Gain34,Gain35,Gain36,Gain37,Gain38,Gain39,Gain40,Gain41,
Gain42,Gain43,Gain44,Gain45,Gain46,Gain47: bit_vector (4 downto
1):="0000";
signal
Gain48,Gain49,Gain50,Gain51,Gain52,Gain53,Gain54,Gain55,Gain56,Gain57,
Gain58,Gain59,Gain60,Gain61,Gain62,Gain63: bit_vector (4 downto
1):="0000";
signal
PhaseInc0,PhaseInc1,PhaseInc2,PhaseInc3,PhaseInc4,PhaseInc5,PhaseInc6,
PhaseInc7,
PhaseInc8,PhaseInc9,PhaseInc10,PhaseInc11,PhaseInc12,PhaseInc13,PhaseIn
c14,PhaseInc15 : bit_vector (5 downto 1):="00000";
Signal
PhaseInc16,PhaseInc17,PhaseInc18,PhaseInc19,PhaseInc20,PhaseInc21,
PhaseInc22,PhaseInc23,
PhaseInc24,PhaseInc25,PhaseInc26,PhaseInc27,PhaseInc28,PhaseInc29,Phase
Inc30,PhaseInc31 : bit_vector (5 downto 1):="00000";
signal
PhaseInc32,PhaseInc33,PhaseInc34,PhaseInc35,PhaseInc36,PhaseInc37,Phase
Inc38,PhaseInc39,
PhaseInc40,PhaseInc41,PhaseInc42,PhaseInc43,PhaseInc44,PhaseInc45,Phase
Inc46,PhaseInc47 : bit_vector (5 downto 1):="00000";
signal
PhaseInc48,PhaseInc49,PhaseInc50,PhaseInc51,PhaseInc52,PhaseInc53,Phase
Inc54,PhaseInc55,
PhaseInc56,PhaseInc57,PhaseInc58,PhaseInc59,PhaseInc60,PhaseInc61,Phase
Inc62,PhaseInc63 : bit_vector (5 downto 1):="00000";
signal
URB0,URB1,URB2,URB3,URB4,URB5,URB6,URB7,URB8,URB9,URB10,URB11,URB12,URB
13,URB14,URB15: bit :='0';
signal
URB16,URB17,URB18,URB19,URB20,URB21,URB22,URB23,URB24,URB25,URB26,URB27
,URB28,URB29,URB30,URB31: bit :='0';
```

```
signal
URB32,URB33,URB34,URB35,URB36,URB37,URB38,URB39,URB40,URB41,URB42,URB43
,URB44,URB45,URB46,URB47: bit :='0';
signal
URB48,URB49,URB50,URB51,URB52,URB53,URB54,URB55,URB56,URB57,URB58,URB59
,URB60,URB61,URB62,URB63: bit :='0';
signal PhaseSamp: bit_vector (5 downto 1):= "00000";
signal UNP, PSVin, ODVin: bit :='0';
signal LD, RESET, PRB, PRB1, PRB2, PRB3, PRB4, PRB5, PRB6, PRB7, PRB8,
PRB9, PRB10, PRB11, PRB12, PRB13, PRB14, PRB15,
PRB1not,PRB2not,PRB3not,PRB4not,PRB5not,PRB6not,PRB7not,PRB8not,PRB9not
,PRB10not,PRB11not,PRB12not,PRB13not,PRB14not,PRB15not: bit;

Begin -- BIN0 is the primary output
--Input: formats, cycle begins when PRB=1, 1 format per clock
-- ten bits of each bin: 10 |  9-5   |4-1 |
--
-- 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
--unused | bin3 | bin2 | bin1 | bin0 |sample |PSVin |UNP |PRB
-- 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
--unused | bin7 | bin6 | bin5 | bin4 |sample |PSVin |UNP |PRB
-- 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
--unused | bin11| bin10| bin9 | bin8 |sample |PSVin |UNP |PRB
-- 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
--unused | bin15| bin14| bin13| bin12|sample |PSVin |UNP |PRB
-- 63-8                             | 7-3   | 2    | 1  | 0
--unused                            |sample |PSVin |UNP |PRB
-- more bins in same pattern up to 63...
-- The first sample can be entered on clock 65 if PSVin and UNP are
taken high
LD<='1';
RESET<='0';
PRB<=Input(0);
UNP<=Input(1);
PSVin<=Input(2);
PhaseSamp<=Input(7 downto 3);
-- Bins 0,4,8,12,16,20,24,28,32,36,40,44,48,52,56,60 data
URB0<=Input(17);
PhaseInc0<=Input(16 downto 12);
Gain0<=Input(11 downto 8);
URB4<=Input(17);
PhaseInc4<=Input(16 downto 12);
Gain4<=Input(11 downto 8);
URB8<=Input(17);
PhaseInc8<=Input(16 downto 12);
Gain8<=Input(11 downto 8);
URB12<=Input(17);
PhaseInc12<=Input(16 downto 12);
Gain12<=Input(11 downto 8);
URB16<=Input(17);
PhaseInc16<=Input(16 downto 12);
Gain16<=Input(11 downto 8);
URB20<=Input(17);
PhaseInc20<=Input(16 downto 12);
Gain20<=Input(11 downto 8);
URB24<=Input(17);
PhaseInc24<=Input(16 downto 12);
```

```
Gain24<=Input(11 downto 8);
URB28<=Input(17);
PhaseInc28<=Input(16 downto 12);
Gain28<=Input(11 downto 8);
URB32<=Input(17);
PhaseInc32<=Input(16 downto 12);
Gain32<=Input(11 downto 8);
URB36<=Input(17);
PhaseInc36<=Input(16 downto 12);
Gain36<=Input(11 downto 8);
URB40<=Input(17);
PhaseInc40<=Input(16 downto 12);
Gain40<=Input(11 downto 8);
URB44<=Input(17);
PhaseInc44<=Input(16 downto 12);
Gain44<=Input(11 downto 8);
URB48<=Input(17);
PhaseInc48<=Input(16 downto 12);
Gain48<=Input(11 downto 8);
URB52<=Input(17);
PhaseInc52<=Input(16 downto 12);
Gain52<=Input(11 downto 8);
URB56<=Input(17);
PhaseInc56<=Input(16 downto 12);
Gain56<=Input(11 downto 8);
URB60<=Input(17);
PhaseInc60<=Input(16 downto 12);
Gain60<=Input(11 downto 8);
-- Bins 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61  data
URB1<=Input(27);
PhaseInc1<=Input(26 downto 22);
Gain1<=Input(21 downto 18);
URB5<=Input(27);
PhaseInc5<=Input(26 downto 22);
Gain5<=Input(21 downto 18);
URB9<=Input(27);
PhaseInc9<=Input(26 downto 22);
Gain9<=Input(21 downto 18);
URB13<=Input(27);
PhaseInc13<=Input(26 downto 22);
Gain13<=Input(21 downto 18);
URB17<=Input(27);
PhaseInc17<=Input(26 downto 22);
Gain17<=Input(21 downto 18);
URB21<=Input(27);
PhaseInc21<=Input(26 downto 22);
Gain21<=Input(21 downto 18);
URB25<=Input(27);
PhaseInc25<=Input(26 downto 22);
Gain25<=Input(21 downto 18);
URB29<=Input(27);
PhaseInc29<=Input(26 downto 22);
Gain29<=Input(21 downto 18);
URB33<=Input(27);
PhaseInc33<=Input(26 downto 22);
Gain33<=Input(21 downto 18);
URB37<=Input(27);
```

```
PhaseInc37<=Input(26 downto 22);
Gain37<=Input(21 downto 18);
URB41<=Input(27);
PhaseInc41<=Input(26 downto 22);
Gain41<=Input(21 downto 18);
URB45<=Input(27);
PhaseInc45<=Input(26 downto 22);
Gain45<=Input(21 downto 18);
URB49<=Input(27);
PhaseInc49<=Input(26 downto 22);
Gain49<=Input(21 downto 18);
URB53<=Input(27);
PhaseInc53<=Input(26 downto 22);
Gain53<=Input(21 downto 18);
URB57<=Input(27);
PhaseInc57<=Input(26 downto 22);
Gain57<=Input(21 downto 18);
URB61<=Input(27);
PhaseInc61<=Input(26 downto 22);
Gain61<=Input(21 downto 18);
-- Bins 2,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62 data
URB2<=Input(37);
PhaseInc2<=Input(36 downto 32);
Gain2<=Input(31 downto 28);
URB6<=Input(37);
PhaseInc6<=Input(36 downto 32);
Gain6<=Input(31 downto 28);
URB10<=Input(37);
PhaseInc10<=Input(36 downto 32);
Gain10<=Input(31 downto 28);
URB14<=Input(37);
PhaseInc14<=Input(36 downto 32);
Gain14<=Input(31 downto 28);
URB18<=Input(37);
PhaseInc18<=Input(36 downto 32);
Gain18<=Input(31 downto 28);
URB22<=Input(37);
PhaseInc22<=Input(36 downto 32);
Gain22<=Input(31 downto 28);
URB26<=Input(37);
PhaseInc26<=Input(36 downto 32);
Gain26<=Input(31 downto 28);
URB30<=Input(37);
PhaseInc30<=Input(36 downto 32);
Gain30<=Input(31 downto 28);
URB34<=Input(37);
PhaseInc34<=Input(36 downto 32);
Gain34<=Input(31 downto 28);
URB38<=Input(37);
PhaseInc38<=Input(36 downto 32);
Gain38<=Input(31 downto 28);
URB42<=Input(37);
PhaseInc42<=Input(36 downto 32);
Gain42<=Input(31 downto 28);
URB46<=Input(37);
PhaseInc46<=Input(36 downto 32);
Gain46<=Input(31 downto 28);
```

```
URB50<=Input(37);
PhaseInc50<=Input(36 downto 32);
Gain50<=Input(31 downto 28);
URB54<=Input(37);
PhaseInc54<=Input(36 downto 32);
Gain54<=Input(31 downto 28);
URB58<=Input(37);
PhaseInc58<=Input(36 downto 32);
Gain58<=Input(31 downto 28);
URB62<=Input(37);
PhaseInc62<=Input(36 downto 32);
Gain62<=Input(31 downto 28);
-- Bins 3,7,11,15,19,23,27,31,35,39,43,47,51,55,59,63 data
URB3<=Input(47);
PhaseInc3<=Input(46 downto 42);
Gain3<=Input(41 downto 38);
URB7<=Input(47);
PhaseInc7<=Input(46 downto 42);
Gain7<=Input(41 downto 38);
URB11<=Input(47);
PhaseInc11<=Input(46 downto 42);
Gain11<=Input(41 downto 38);
URB15<=Input(47);
PhaseInc15<=Input(46 downto 42);
Gain15<=Input(41 downto 38);
URB19<=Input(47);
PhaseInc19<=Input(46 downto 42);
Gain19<=Input(41 downto 38);
URB23<=Input(47);
PhaseInc23<=Input(46 downto 42);
Gain23<=Input(41 downto 38);
URB27<=Input(47);
PhaseInc27<=Input(46 downto 42);
Gain27<=Input(41 downto 38);
URB31<=Input(47);
PhaseInc31<=Input(46 downto 42);
Gain31<=Input(41 downto 38);
URB35<=Input(47);
PhaseInc35<=Input(46 downto 42);
Gain35<=Input(41 downto 38);
URB39<=Input(47);
PhaseInc39<=Input(46 downto 42);
Gain39<=Input(41 downto 38);
URB43<=Input(47);
PhaseInc43<=Input(46 downto 42);
Gain43<=Input(41 downto 38);
URB47<=Input(47);
PhaseInc47<=Input(46 downto 42);
Gain47<=Input(41 downto 38);
URB51<=Input(47);
PhaseInc51<=Input(46 downto 42);
Gain51<=Input(41 downto 38);
URB55<=Input(47);
PhaseInc55<=Input(46 downto 42);
Gain55<=Input(41 downto 38);
URB59<=Input(47);
PhaseInc59<=Input(46 downto 42);
```

```
Gain59<=Input(41 downto 38);
URB63<=Input(47);
PhaseInc63<=Input(46 downto 42);
Gain63<=Input(41 downto 38);

DFF0: DFlipFlop port map (CLK, LD, RESET, PRB, PRB1, PRB1not);
DFF1: DFlipFlop port map (CLK, LD, RESET, PRB1, PRB2, PRB2not);
DFF2: DFlipFlop port map (CLK, LD, RESET, PRB2, PRB3, PRB3not);
DFF3: DFlipFlop port map (CLK, LD, RESET, PRB3, PRB4, PRB4not);
DFF4: DFlipFlop port map (CLK, LD, RESET, PRB4, PRB5, PRB5not);
DFF5: DFlipFlop port map (CLK, LD, RESET, PRB5, PRB6, PRB6not);
DFF6: DFlipFlop port map (CLK, LD, RESET, PRB6, PRB7, PRB7not);
DFF7: DFlipFlop port map (CLK, LD, RESET, PRB7, PRB8, PRB8not);
DFF8: DFlipFlop port map (CLK, LD, RESET, PRB8, PRB9, PRB9not);
DFF9: DFlipFlop port map (CLK, LD, RESET, PRB9, PRB10, PRB10not);
DFF10: DFlipFlop port map (CLK, LD, RESET, PRB10, PRB11, PRB11not);
DFF11: DFlipFlop port map (CLK, LD, RESET, PRB11, PRB12, PRB12not);
DFF12: DFlipFlop port map (CLK, LD, RESET, PRB12, PRB13, PRB13not);
DFF13: DFlipFlop port map (CLK, LD, RESET, PRB13, PRB14, PRB14not);
DFF14: DFlipFlop port map (CLK, LD, RESET, PRB14, PRB15, PRB15not);

BIN0: OneBin port map (DRFM1, PhaseInc0, Gain0, URB0, UNP, PRB,
ODVout1, PSVout1, Q1, I1, Q, I, ODVout0, PSVout0, DRFM0, CLK);
BIN1: OneBin port map (DRFM2, PhaseInc1, Gain1, URB1, UNP, PRB,
ODVout2, PSVout2, Q2, I2, Q1, I1, ODVout1, PSVout1, DRFM1, CLK);
BIN2: OneBin port map (DRFM3, PhaseInc2, Gain2, URB2, UNP, PRB,
ODVout3, PSVout3, Q3, I3, Q2, I2, ODVout2, PSVout2, DRFM2, CLK);
BIN3: OneBin port map (DRFM4, PhaseInc3, Gain3, URB3, UNP, PRB,
ODVout4, PSVout4, Q4, I4, Q3, I3, ODVout3, PSVout3, DRFM3, CLK);
BIN4: OneBin port map (DRFM5, PhaseInc4, Gain4, URB4, UNP, PRB1,
ODVout5, PSVout5, Q5, I5, Q4, I4, ODVout4, PSVout4, DRFM4, CLK);
BIN5: OneBin port map (DRFM6, PhaseInc5, Gain5, URB5, UNP, PRB1,
ODVout6, PSVout6, Q6, I6, Q5, I5, ODVout5, PSVout5, DRFM5, CLK);
BIN6: OneBin port map (DRFM7, PhaseInc6, Gain6, URB6, UNP, PRB1,
ODVout7, PSVout7, Q7, I7, Q6, I6, ODVout6, PSVout6, DRFM6, CLK);
BIN7: OneBin port map (DRFM8, PhaseInc7, Gain7, URB7, UNP, PRB1,
ODVout8, PSVout8, Q8, I8, Q7, I7, ODVout7, PSVout7, DRFM7, CLK);
BIN8: OneBin port map (DRFM9, PhaseInc8, Gain8, URB8, UNP, PRB2,
ODVout9, PSVout9, Q9, I9, Q8, I8, ODVout8, PSVout8, DRFM8, CLK);
BIN9: OneBin port map (DRFM10, PhaseInc9, Gain9, URB9, UNP, PRB2,
ODVout10, PSVout10, Q10, I10, Q9, I9, ODVout9, PSVout9, DRFM9, CLK);
BIN10: OneBin port map (DRFM11, PhaseInc10, Gain10, URB10, UNP, PRB2,
ODVout11, PSVout11, Q11, I11, Q10, I10, ODVout10, PSVout10, DRFM10,
CLK);
BIN11: OneBin port map (DRFM12, PhaseInc11, Gain11, URB11, UNP, PRB2,
ODVout12, PSVout12, Q12, I12, Q11, I11, ODVout11, PSVout11, DRFM11,
CLK);
BIN12: OneBin port map (DRFM13, PhaseInc12, Gain12, URB12, UNP, PRB3,
ODVout13, PSVout13, Q13, I13, Q12, I12, ODVout12, PSVout12, DRFM12,
CLK);
BIN13: OneBin port map (DRFM14, PhaseInc13, Gain13, URB13, UNP, PRB3,
ODVout14, PSVout14, Q14, I14, Q13, I13, ODVout13, PSVout13, DRFM13,
CLK);
BIN14: OneBin port map (DRFM15, PhaseInc14, Gain14, URB14, UNP, PRB3,
ODVout15, PSVout15, Q15, I15, Q14, I14, ODVout14, PSVout14, DRFM14,
CLK);
```

```
BIN15: OneBin port map (DRFM16, PhaseInc15, Gain15, URB15, UNP, PRB3,
ODVout16, PSVout16, Q16, I16, Q15, I15, ODVout15, PSVout15, DRFM15,
CLK);
BIN16: OneBin port map (DRFM17, PhaseInc16, Gain16, URB16, UNP, PRB4,
ODVout17, PSVout17, Q17, I17, Q16, I16, ODVout16, PSVout16, DRFM16,
CLK);
BIN17: OneBin port map (DRFM18, PhaseInc17, Gain17, URB17, UNP, PRB4,
ODVout18, PSVout18, Q18, I18, Q17, I17, ODVout17, PSVout17, DRFM17,
CLK);
BIN18: OneBin port map (DRFM19, PhaseInc18, Gain18, URB18, UNP, PRB4,
ODVout19, PSVout19, Q19, I19, Q18, I18, ODVout18, PSVout18, DRFM18,
CLK);
BIN19: OneBin port map (DRFM20, PhaseInc19, Gain19, URB19, UNP, PRB4,
ODVout20, PSVout20, Q20, I20, Q19, I19, ODVout19, PSVout19, DRFM19,
CLK);
BIN20: OneBin port map (DRFM21, PhaseInc20, Gain20, URB20, UNP, PRB5,
ODVout21, PSVout21, Q21, I21, Q20, I20, ODVout20, PSVout20, DRFM20,
CLK);
BIN21: OneBin port map (DRFM22, PhaseInc21, Gain21, URB21, UNP, PRB5,
ODVout22, PSVout22, Q22, I22, Q21, I21, ODVout21, PSVout21, DRFM21,
CLK);
BIN22: OneBin port map (DRFM23, PhaseInc22, Gain22, URB22, UNP, PRB5,
ODVout23, PSVout23, Q23, I23, Q22, I22, ODVout22, PSVout22, DRFM22,
CLK);
BIN23: OneBin port map (DRFM24, PhaseInc23, Gain23, URB23, UNP, PRB5,
ODVout24, PSVout24, Q24, I24, Q23, I23, ODVout23, PSVout23, DRFM23,
CLK);
BIN24: OneBin port map (DRFM25, PhaseInc24, Gain24, URB24, UNP, PRB6,
ODVout25, PSVout25, Q25, I25, Q24, I24, ODVout24, PSVout24, DRFM24,
CLK);
BIN25: OneBin port map (DRFM26, PhaseInc25, Gain25, URB25, UNP, PRB6,
ODVout26, PSVout26, Q26, I26, Q25, I25, ODVout25, PSVout25, DRFM25,
CLK);
BIN26: OneBin port map (DRFM27, PhaseInc26, Gain26, URB26, UNP, PRB6,
ODVout27, PSVout27, Q27, I27, Q26, I26, ODVout26, PSVout26, DRFM26,
CLK);
BIN27: OneBin port map (DRFM28, PhaseInc27, Gain27, URB27, UNP, PRB6,
ODVout28, PSVout28, Q28, I28, Q27, I27, ODVout27, PSVout27, DRFM27,
CLK);
BIN28: OneBin port map (DRFM29, PhaseInc28, Gain28, URB28, UNP, PRB7,
ODVout29, PSVout29, Q29, I29, Q28, I28, ODVout28, PSVout28, DRFM28,
CLK);
BIN29: OneBin port map (DRFM30, PhaseInc29, Gain29, URB29, UNP, PRB7,
ODVout30, PSVout30, Q30, I30, Q29, I29, ODVout29, PSVout29, DRFM29,
CLK);
BIN30: OneBin port map (DRFM31, PhaseInc30, Gain30, URB30, UNP, PRB7,
ODVout31, PSVout31, Q31, I31, Q30, I30, ODVout30, PSVout30, DRFM30,
CLK);
BIN31: OneBin port map (DRFM32, PhaseInc31, Gain31, URB31, UNP, PRB7,
ODVout32, PSVout32, Q32, I32, Q31, I31, ODVout31, PSVout31, DRFM31,
CLK);
BIN32: OneBin port map (DRFM33, PhaseInc32, Gain32, URB32, UNP, PRB8,
ODVout33, PSVout33, Q33, I33, Q32, I32, ODVout32, PSVout32, DRFM32,
CLK);
BIN33: OneBin port map (DRFM34, PhaseInc33, Gain33, URB33, UNP, PRB8,
ODVout34, PSVout34, Q34, I34, Q33, I33, ODVout33, PSVout33, DRFM33,
CLK);
```

```
BIN34: OneBin port map (DRFM35, PhaseInc34, Gain34, URB34, UNP, PRB8,
ODVout35, PSVout35, Q35, I35, Q34, I34, ODVout34, PSVout34, DRFM34,
CLK);
BIN35: OneBin port map (DRFM36, PhaseInc35, Gain35, URB35, UNP, PRB8,
ODVout36, PSVout36, Q36, I36, Q35, I35, ODVout35, PSVout35, DRFM35,
CLK);
BIN36: OneBin port map (DRFM37, PhaseInc36, Gain36, URB36, UNP, PRB9,
ODVout37, PSVout37, Q37, I37, Q36, I36, ODVout36, PSVout36, DRFM36,
CLK);
BIN37: OneBin port map (DRFM38, PhaseInc37, Gain37, URB37, UNP, PRB9,
ODVout38, PSVout38, Q38, I38, Q37, I37, ODVout37, PSVout37, DRFM37,
CLK);
BIN38: OneBin port map (DRFM39, PhaseInc38, Gain38, URB38, UNP, PRB9,
ODVout39, PSVout39, Q39, I39, Q38, I38, ODVout38, PSVout38, DRFM38,
CLK);
BIN39: OneBin port map (DRFM40, PhaseInc39, Gain39, URB39, UNP, PRB9,
ODVout40, PSVout40, Q40, I40, Q39, I39, ODVout39, PSVout39, DRFM39,
CLK);
BIN40: OneBin port map (DRFM41, PhaseInc40, Gain40, URB40, UNP, PRB10,
ODVout41, PSVout41, Q41, I41, Q40, I40, ODVout40, PSVout40, DRFM40,
CLK);
BIN41: OneBin port map (DRFM42, PhaseInc41, Gain41, URB41, UNP, PRB10,
ODVout42, PSVout42, Q42, I42, Q41, I41, ODVout41, PSVout41, DRFM41,
CLK);
BIN42: OneBin port map (DRFM43, PhaseInc42, Gain42, URB42, UNP, PRB10,
ODVout43, PSVout43, Q43, I43, Q42, I42, ODVout42, PSVout42, DRFM42,
CLK);
BIN43: OneBin port map (DRFM44, PhaseInc43, Gain43, URB43, UNP, PRB10,
ODVout44, PSVout44, Q44, I44, Q43, I43, ODVout43, PSVout43, DRFM43,
CLK);
BIN44: OneBin port map (DRFM45, PhaseInc44, Gain44, URB44, UNP, PRB11,
ODVout45, PSVout45, Q45, I45, Q44, I44, ODVout44, PSVout44, DRFM44,
CLK);
BIN45: OneBin port map (DRFM46, PhaseInc45, Gain45, URB45, UNP, PRB11,
ODVout46, PSVout46, Q46, I46, Q45, I45, ODVout45, PSVout45, DRFM45,
CLK);
BIN46: OneBin port map (DRFM47, PhaseInc46, Gain46, URB46, UNP, PRB11,
ODVout47, PSVout47, Q47, I47, Q46, I46, ODVout46, PSVout46, DRFM46,
CLK);
BIN47: OneBin port map (DRFM48, PhaseInc47, Gain47, URB47, UNP, PRB11,
ODVout48, PSVout48, Q48, I48, Q47, I47, ODVout47, PSVout47, DRFM47,
CLK);
BIN48: OneBin port map (DRFM49, PhaseInc48, Gain48, URB48, UNP, PRB12,
ODVout49, PSVout49, Q49, I49, Q48, I48, ODVout48, PSVout48, DRFM48,
CLK);
BIN49: OneBin port map (DRFM50, PhaseInc49, Gain49, URB49, UNP, PRB12,
ODVout50, PSVout50, Q50, I50, Q49, I49, ODVout49, PSVout49, DRFM49,
CLK);
BIN50: OneBin port map (DRFM51, PhaseInc50, Gain50, URB50, UNP, PRB12,
ODVout51, PSVout51, Q51, I51, Q50, I50, ODVout50, PSVout50, DRFM50,
CLK);
BIN51: OneBin port map (DRFM52, PhaseInc51, Gain51, URB51, UNP, PRB12,
ODVout52, PSVout52, Q52, I52, Q51, I51, ODVout51, PSVout51, DRFM51,
CLK);
BIN52: OneBin port map (DRFM53, PhaseInc52, Gain52, URB52, UNP, PRB13,
ODVout53, PSVout53, Q53, I53, Q52, I52, ODVout52, PSVout52, DRFM52,
CLK);
```

```
BIN53: OneBin port map (DRFM54, PhaseInc53, Gain53, URB53, UNP, PRB13,
ODVout54, PSVout54, Q54, I54, Q53, I53, ODVout53, PSVout53, DRFM53,
CLK);
BIN54: OneBin port map (DRFM55, PhaseInc54, Gain54, URB54, UNP, PRB13,
ODVout55, PSVout55, Q55, I55, Q54, I54, ODVout54, PSVout54, DRFM54,
CLK);
BIN55: OneBin port map (DRFM56, PhaseInc55, Gain55, URB55, UNP, PRB13,
ODVout56, PSVout56, Q56, I56, Q55, I55, ODVout55, PSVout55, DRFM55,
CLK);
BIN56: OneBin port map (DRFM57, PhaseInc56, Gain56, URB56, UNP, PRB14,
ODVout57, PSVout57, Q57, I57, Q56, I56, ODVout56, PSVout56, DRFM56,
CLK);
BIN57: OneBin port map (DRFM58, PhaseInc57, Gain57, URB57, UNP, PRB14,
ODVout58, PSVout58, Q58, I58, Q57, I57, ODVout57, PSVout57, DRFM57,
CLK);
BIN58: OneBin port map (DRFM59, PhaseInc58, Gain58, URB58, UNP, PRB14,
ODVout59, PSVout59, Q59, I59, Q58, I58, ODVout58, PSVout58, DRFM58,
CLK);
BIN59: OneBin port map (DRFM60, PhaseInc59, Gain59, URB59, UNP, PRB14,
ODVout60, PSVout60, Q60, I60, Q59, I59, ODVout59, PSVout59, DRFM59,
CLK);
BIN60: OneBin port map (DRFM61, PhaseInc60, Gain60, URB60, UNP, PRB15,
ODVout61, PSVout61, Q61, I61, Q60, I60, ODVout60, PSVout60, DRFM60,
CLK);
BIN61: OneBin port map (DRFM62, PhaseInc61, Gain61, URB61, UNP, PRB15,
ODVout62, PSVout62, Q62, I62, Q61, I61, ODVout61, PSVout61, DRFM61,
CLK);
BIN62: OneBin port map (DRFM63, PhaseInc62, Gain62, URB62, UNP, PRB15,
ODVout63, PSVout63, Q63, I63, Q62, I62, ODVout62, PSVout62, DRFM62,
CLK);
BIN63: OneBin port map (PhaseSamp, PhaseInc63, Gain63, URB63, UNP,
PRB15, ODVin, PSVin, "00000000000000000", "00000000000000000", Q63,
I63,ODVout63, PSVout63, DRFM63, CLK);

-- Output Format:
-- 63-41 |   40   |   39   | 38-22 | 21-5 | 4-0
--unused | PSVout | ODVout |   Q   |   I  | DRFM

Output(40)<=PSVout0;
Output(39)<=ODVout0;
Output(38 downto 22)<=Q;
Output(21 downto 5)<=I;
Output(4 downto 0)<=DRFM0;
Output(63 downto 41)<="00000000000000000000000";

end SixtyFourBin;
```

E. 128 BIN MACRO VHDL

```
entity One28Bin is
port (Input: in bit_vector (63 downto 0);
Output:out bit_vector (63 downto 0); CLK: in bit);
end One28Bin;

architecture One28Bin of One28Bin is

component OneBin is
```

```
port (PhaseSamp, PhaseInc: in bit_vector (5 downto 1);
Gain: in bit_vector (4 downto 1);
URB, UNP, PRB, ODVin, PSVin: in bit;
OtherBinDataSIN,OtherBinDataCOS:in bit_vector (17 downto 1);
Q, I: out bit_vector (17 downto 1);
ODVout, PSVout: out bit;
DRFM: out bit_vector (5 downto 1);
CLK: in bit);
end component;
component DFlipFlop
port (CLK, LD, RESET, D: in bit;
Q: inout bit; Qnot: out bit);
end component;

signal Q, I, Q1,I1,Q2,I2,Q3,I3,Q4,I4,Q5,I5,Q6,I6,Q7,I7,Q8,I8,Q9,I9,
Q10,I10,Q11,I11,Q12,I12,Q13,I13,Q14,I14,Q15,I15: bit_vector (17 downto
1):="00000000000000000";
signal Q16, I16,
Q17,I17,Q18,I18,Q19,I19,Q20,I20,Q21,I21,Q22,I22,Q23,I23,Q24,I24,Q25,I25
,
Q26,I26,Q27,I27,Q28,I28,Q29,I29,Q30,I30,Q31,I31: bit_vector (17 downto
1):="00000000000000000";
signal Q32, I32,
Q33,I33,Q34,I34,Q35,I35,Q36,I36,Q37,I37,Q38,I38,Q39,I39,Q40,I40,Q41,I41
,
Q42,I42,Q43,I43,Q44,I44,Q45,I45,Q46,I46,Q47,I47: bit_vector (17 downto
1):="00000000000000000";
signal Q48, I48,
Q49,I49,Q50,I50,Q51,I51,Q52,I52,Q53,I53,Q54,I54,Q55,I55,Q56,I56,Q57,I57
,
Q58,I58,Q59,I59,Q60,I60,Q61,I61,Q62,I62,Q63,I63: bit_vector (17 downto
1):="00000000000000000";
signal Q64, I64,
Q65,I65,Q66,I66,Q67,I67,Q68,I68,Q69,I69,Q70,I70,Q71,I71,Q72,I72,Q73,I73
,
Q74,I74,Q75,I75,Q76,I76,Q77,I77,Q78,I78,Q79,I79: bit_vector (17 downto
1):="00000000000000000";
signal Q80, I80,
Q81,I81,Q82,I82,Q83,I83,Q84,I84,Q85,I85,Q86,I86,Q87,I87,Q88,I88,Q89,I89
,
Q90,I90,Q91,I91,Q92,I92,Q93,I93,Q94,I94,Q95,I95: bit_vector (17 downto
1):="00000000000000000";
signal Q96, I96,
Q97,I97,Q98,I98,Q99,I99,Q100,I100,Q101,I101,Q102,I102,Q103,I103,Q104,I1
04,Q105,I105,
Q106,I106,Q107,I107,Q108,I108,Q109,I109,Q110,I110,Q111,I111: bit_vector
(17 downto 1):="00000000000000000";
signal Q112, I112,
Q113,I113,Q114,I114,Q115,I115,Q116,I116,Q117,I117,Q118,I118,Q119,I119,Q
120,I120,Q121,I121,
Q122,I122,Q123,I123,Q124,I124,Q125,I125,Q126,I126,Q127,I127: bit_vector
(17 downto 1):="00000000000000000";

signal DRFM0,DRFM1,DRFM2,DRFM3,DRFM4,DRFM5,DRFM6,DRFM7,DRFM8,DRFM9,
DRFM10,DRFM11,DRFM12,DRFM13,DRFM14,DRFM15: bit_vector (5 downto
1):="00000";
```

```
signal
DRFM16,DRFM17,DRFM18,DRFM19,DRFM20,DRFM21,DRFM22,DRFM23,DRFM24,DRFM25,
DRFM26,DRFM27,DRFM28,DRFM29,DRFM30,DRFM31: bit_vector (5 downto
1):="00000";
signal
DRFM32,DRFM33,DRFM34,DRFM35,DRFM36,DRFM37,DRFM38,DRFM39,DRFM40,DRFM41,
DRFM42,DRFM43,DRFM44,DRFM45,DRFM46,DRFM47: bit_vector (5 downto
1):="00000";
signal
DRFM48,DRFM49,DRFM50,DRFM51,DRFM52,DRFM53,DRFM54,DRFM55,DRFM56,DRFM57,
DRFM58,DRFM59,DRFM60,DRFM61,DRFM62,DRFM63: bit_vector (5 downto
1):="00000";
signal
DRFM64,DRFM65,DRFM66,DRFM67,DRFM68,DRFM69,DRFM70,DRFM71,DRFM72,DRFM73,
DRFM74,DRFM75,DRFM76,DRFM77,DRFM78,DRFM79: bit_vector (5 downto
1):="00000";
signal
DRFM80,DRFM81,DRFM82,DRFM83,DRFM84,DRFM85,DRFM86,DRFM87,DRFM88,DRFM89,
DRFM90,DRFM91,DRFM92,DRFM93,DRFM94,DRFM95: bit_vector (5 downto
1):="00000";
signal
DRFM96,DRFM97,DRFM98,DRFM99,DRFM100,DRFM101,DRFM102,DRFM103,DRFM104,DRF
M105,
DRFM106,DRFM107,DRFM108,DRFM109,DRFM110,DRFM111: bit_vector (5 downto
1):="00000";
signal
DRFM112,DRFM113,DRFM114,DRFM115,DRFM116,DRFM117,DRFM118,DRFM119,DRFM120
,DRFM121,
DRFM122,DRFM123,DRFM124,DRFM125,DRFM126,DRFM127: bit_vector (5 downto
1):="00000";

signal PSVout0,PSVout1,PSVout2,PSVout3,PSVout4,PSVout5,PSVout6,
PSVout7,PSVout8,PSVout9,PSVout10,PSVout11,PSVout12,PSVout13,PSVout14,PS
Vout15: bit:='0';
signal PSVout16,PSVout17,PSVout18,PSVout19,PSVout20,PSVout21,PSVout22,
PSVout23,PSVout24,PSVout25,PSVout26,PSVout27,PSVout28,PSVout29,PSVout30
,PSVout31: bit:='0';
signal PSVout32,PSVout33,PSVout34,PSVout35,PSVout36,PSVout37,PSVout38,
PSVout39,PSVout40,PSVout41,PSVout42,PSVout43,PSVout44,PSVout45,PSVout46
,PSVout47: bit:='0';
signal PSVout48,PSVout49,PSVout50,PSVout51,PSVout52,PSVout53,PSVout54,
PSVout55,PSVout56,PSVout57,PSVout58,PSVout59,PSVout60,PSVout61,PSVout62
,PSVout63: bit:='0';
signal PSVout64,PSVout65,PSVout66,PSVout67,PSVout68,PSVout69,PSVout70,
PSVout71,PSVout72,PSVout73,PSVout74,PSVout75,PSVout76,PSVout77,PSVout78
,PSVout79: bit:='0';
signal PSVout80,PSVout81,PSVout82,PSVout83,PSVout84,PSVout85,PSVout86,
PSVout87,PSVout88,PSVout89,PSVout90,PSVout91,PSVout92,PSVout93,PSVout94
,PSVout95: bit:='0';
signal
PSVout96,PSVout97,PSVout98,PSVout99,PSVout100,PSVout101,PSVout102,
PSVout103,PSVout104,PSVout105,PSVout106,PSVout107,PSVout108,PSVout109,P
SVout110,PSVout111: bit:='0';
signal
PSVout112,PSVout113,PSVout114,PSVout115,PSVout116,PSVout117,PSVout118,
PSVout119,PSVout120,PSVout121,PSVout122,PSVout123,PSVout124,PSVout125,P
SVout126,PSVout127: bit:='0';
```

```
signal ODVout0,ODVout1,ODVout2,ODVout3,ODVout4,ODVout5,ODVout6,
ODVout7,ODVout8,ODVout9,ODVout10,ODVout11,ODVout12,ODVout13,ODVout14,OD
Vout15: bit:='0';
signal ODVout16,ODVout17,ODVout18,ODVout19,ODVout20,ODVout21,ODVout22,
ODVout23,ODVout24,ODVout25,ODVout26,ODVout27,ODVout28,ODVout29,ODVout30
,ODVout31: bit:='0';
signal ODVout32,ODVout33,ODVout34,ODVout35,ODVout36,ODVout37,ODVout38,
ODVout39,ODVout40,ODVout41,ODVout42,ODVout43,ODVout44,ODVout45,ODVout46
,ODVout47: bit:='0';
signal ODVout48,ODVout49,ODVout50,ODVout51,ODVout52,ODVout53,ODVout54,
ODVout55,ODVout56,ODVout57,ODVout58,ODVout59,ODVout60,ODVout61,ODVout62
,ODVout63: bit:='0';
signal ODVout64,ODVout65,ODVout66,ODVout67,ODVout68,ODVout69,ODVout70,
ODVout71,ODVout72,ODVout73,ODVout74,ODVout75,ODVout76,ODVout77,ODVout78
,ODVout79: bit:='0';
signal ODVout80,ODVout81,ODVout82,ODVout83,ODVout84,ODVout85,ODVout86,
ODVout87,ODVout88,ODVout89,ODVout90,ODVout91,ODVout92,ODVout93,ODVout94
,ODVout95: bit:='0';
signal
ODVout96,ODVout97,ODVout98,ODVout99,ODVout100,ODVout101,ODVout102,
ODVout103,ODVout104,ODVout105,ODVout106,ODVout107,ODVout108,ODVout109,O
DVout110,ODVout111: bit:='0';
signal
ODVout112,ODVout113,ODVout114,ODVout115,ODVout116,ODVout117,ODVout118,
ODVout119,ODVout120,ODVout121,ODVout122,ODVout123,ODVout124,ODVout125,O
DVout126,ODVout127: bit:='0';

signal Gain0,Gain1,Gain2,Gain3,Gain4,Gain5,Gain6,Gain7,Gain8,Gain9,
Gain10,Gain11,Gain12,Gain13,Gain14,Gain15: bit_vector (4 downto
1):="0000";
signal
Gain16,Gain17,Gain18,Gain19,Gain20,Gain21,Gain22,Gain23,Gain24,Gain25,
Gain26,Gain27,Gain28,Gain29,Gain30,Gain31: bit_vector (4 downto
1):="0000";
signal
Gain32,Gain33,Gain34,Gain35,Gain36,Gain37,Gain38,Gain39,Gain40,Gain41,
Gain42,Gain43,Gain44,Gain45,Gain46,Gain47: bit_vector (4 downto
1):="0000";
signal
Gain48,Gain49,Gain50,Gain51,Gain52,Gain53,Gain54,Gain55,Gain56,Gain57,
Gain58,Gain59,Gain60,Gain61,Gain62,Gain63: bit_vector (4 downto
1):="0000";
signal
Gain64,Gain65,Gain66,Gain67,Gain68,Gain69,Gain70,Gain71,Gain72,Gain73,
Gain74,Gain75,Gain76,Gain77,Gain78,Gain79: bit_vector (4 downto
1):="0000";
signal
Gain80,Gain81,Gain82,Gain83,Gain84,Gain85,Gain86,Gain87,Gain88,Gain89,
Gain90,Gain91,Gain92,Gain93,Gain94,Gain95: bit_vector (4 downto
1):="0000";
signal
Gain96,Gain97,Gain98,Gain99,Gain100,Gain101,Gain102,Gain103,Gain104,Gai
n105,
Gain106,Gain107,Gain108,Gain109,Gain110,Gain111: bit_vector (4 downto
1):="0000";
```

```
signal
Gain112,Gain113,Gain114,Gain115,Gain116,Gain117,Gain118,Gain119,Gain120
,Gain121,
Gain122,Gain123,Gain124,Gain125,Gain126,Gain127: bit_vector (4 downto
1):="0000";

signal
PhaseInc0,PhaseInc1,PhaseInc2,PhaseInc3,PhaseInc4,PhaseInc5,PhaseInc6,P
haseInc7,
PhaseInc8,PhaseInc9,PhaseInc10,PhaseInc11,PhaseInc12,PhaseInc13,PhaseIn
c14,PhaseInc15 : bit_vector (5 downto 1):="00000";
signal
PhaseInc16,PhaseInc17,PhaseInc18,PhaseInc19,PhaseInc20,PhaseInc21,Phase
Inc22,PhaseInc23,
PhaseInc24,PhaseInc25,PhaseInc26,PhaseInc27,PhaseInc28,PhaseInc29,Phase
Inc30,PhaseInc31 : bit_vector (5 downto 1):="00000";
signal
PhaseInc32,PhaseInc33,PhaseInc34,PhaseInc35,PhaseInc36,PhaseInc37,Phase
Inc38,PhaseInc39,
PhaseInc40,PhaseInc41,PhaseInc42,PhaseInc43,PhaseInc44,PhaseInc45,Phase
Inc46,PhaseInc47 : bit_vector (5 downto 1):="00000";
signal
PhaseInc48,PhaseInc49,PhaseInc50,PhaseInc51,PhaseInc52,PhaseInc53,Phase
Inc54,PhaseInc55,
PhaseInc56,PhaseInc57,PhaseInc58,PhaseInc59,PhaseInc60,PhaseInc61,Phase
Inc62,PhaseInc63 : bit_vector (5 downto 1):="00000";
signal
PhaseInc64,PhaseInc65,PhaseInc66,PhaseInc67,PhaseInc68,PhaseInc69,Phase
Inc70,PhaseInc71,
PhaseInc72,PhaseInc73,PhaseInc74,PhaseInc75,PhaseInc76,PhaseInc77,Phase
Inc78,PhaseInc79 : bit_vector (5 downto 1):="00000";
signal
PhaseInc80,PhaseInc81,PhaseInc82,PhaseInc83,PhaseInc84,PhaseInc85,Phase
Inc86,PhaseInc87,
PhaseInc88,PhaseInc89,PhaseInc90,PhaseInc91,PhaseInc92,PhaseInc93,Phase
Inc94,PhaseInc95 : bit_vector (5 downto 1):="00000";
signal
PhaseInc96,PhaseInc97,PhaseInc98,PhaseInc99,PhaseInc100,PhaseInc101,Pha
seInc102,PhaseInc103,
PhaseInc104,PhaseInc105,PhaseInc106,PhaseInc107,PhaseInc108,PhaseInc109
,PhaseInc110,PhaseInc111 : bit_vector (5 downto 1):="00000";
signal
PhaseInc112,PhaseInc113,PhaseInc114,PhaseInc115,PhaseInc116,PhaseInc117
,PhaseInc118,PhaseInc119,
PhaseInc120,PhaseInc121,PhaseInc122,PhaseInc123,PhaseInc124,PhaseInc125
,PhaseInc126,PhaseInc127 : bit_vector (5 downto 1):="00000";

signal
URB0,URB1,URB2,URB3,URB4,URB5,URB6,URB7,URB8,URB9,URB10,URB11,URB12,URB
13,URB14,URB15: bit :='0';
signal
URB16,URB17,URB18,URB19,URB20,URB21,URB22,URB23,URB24,URB25,URB26,URB27
,URB28,URB29,URB30,URB31: bit :='0';
signal
URB32,URB33,URB34,URB35,URB36,URB37,URB38,URB39,URB40,URB41,URB42,URB43
,URB44,URB45,URB46,URB47: bit :='0';
```

```
signal
URB48,URB49,URB50,URB51,URB52,URB53,URB54,URB55,URB56,URB57,URB58,URB59
,URB60,URB61,URB62,URB63: bit :='0';
signal
URB64,URB65,URB66,URB67,URB68,URB69,URB70,URB71,URB72,URB73,URB74,URB75
,URB76,URB77,URB78,URB79: bit :='0';
signal
URB80,URB81,URB82,URB83,URB84,URB85,URB86,URB87,URB88,URB89,URB90,URB91
,URB92,URB93,URB94,URB95: bit :='0';
signal
URB96,URB97,URB98,URB99,URB100,URB101,URB102,URB103,URB104,URB105,URB10
6,URB107,URB108,URB109,URB110,URB111: bit :='0';
signal
URB112,URB113,URB114,URB115,URB116,URB117,URB118,URB119,URB120,URB121,U
RB122,URB123,URB124,URB125,URB126,URB127: bit :='0';

signal PhaseSamp: bit_vector (5 downto 1):= "00000";
signal UNP, PSVin, ODVin: bit :='0';
signal LD, RESET, PRB, PRB1, PRB2, PRB3, PRB4, PRB5, PRB6, PRB7, PRB8,
PRB9, PRB10, PRB11, PRB12, PRB13, PRB14, PRB15,
PRB16, PRB17, PRB18, PRB19, PRB20, PRB21, PRB22, PRB23, PRB24, PRB25,
PRB26, PRB27, PRB28, PRB29, PRB30, PRB31,
PRB1not,PRB2not,PRB3not,PRB4not,PRB5not,PRB6not,PRB7not,PRB8not,PRB9not
,PRB10not,PRB11not,PRB12not,PRB13not,PRB14not,PRB15not,
PRB16not,PRB17not,PRB18not,PRB19not,PRB20not,PRB21not,PRB22not,PRB23not
,PRB24not,PRB25not,PRB26not,PRB27not,
PRB28not,PRB29not,PRB30not,PRB31not:bit;

Begin -- BIN0 is the primary output
--Input: formats, cycle begins when PRB=1, 1 format per clock
-- ten bits of each bin: 10 |  9-5   |4-1 |
--
-- 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
--unused | bin3 | bin2 | bin1 | bin0 |sample |PSVin |UNP |PRB
-- 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
--unused | bin7 | bin6 | bin5 | bin4 |sample |PSVin |UNP |PRB
-- 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
--unused | bin11| bin10| bin9 | bin8 |sample |PSVin |UNP |PRB
-- 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
--unused | bin15| bin14| bin13| bin12|sample |PSVin |UNP |PRB
-- 63-8                              | 7-3   | 2    | 1  | 0
--unused                             |sample |PSVin |UNP |PRB
-- more bins in same pattern up to 128...
-- The first sample can be entered on clock 129 if PSVin and UNP are
taken high
LD<='1';
RESET<='0';
PRB<=Input(0);
UNP<=Input(1);
PSVin<=Input(2);
PhaseSamp<=Input(7 downto 3);
-- Bins 0,4,8,12,16,20,24,28,32,36,40,44,48,52,56,60...124 data
URB0<=Input(17);
PhaseInc0<=Input(16 downto 12);
Gain0<=Input(11 downto 8);
URB4<=Input(17);
PhaseInc4<=Input(16 downto 12);
```

```
Gain4<=Input(11 downto 8);
URB8<=Input(17);
PhaseInc8<=Input(16 downto 12);
Gain8<=Input(11 downto 8);
URB12<=Input(17);
PhaseInc12<=Input(16 downto 12);
Gain12<=Input(11 downto 8);
URB16<=Input(17);
PhaseInc16<=Input(16 downto 12);
Gain16<=Input(11 downto 8);
URB20<=Input(17);
PhaseInc20<=Input(16 downto 12);
Gain20<=Input(11 downto 8);
URB24<=Input(17);
PhaseInc24<=Input(16 downto 12);
Gain24<=Input(11 downto 8);
URB28<=Input(17);
PhaseInc28<=Input(16 downto 12);
Gain28<=Input(11 downto 8);
URB32<=Input(17);
PhaseInc32<=Input(16 downto 12);
Gain32<=Input(11 downto 8);
URB36<=Input(17);
PhaseInc36<=Input(16 downto 12);
Gain36<=Input(11 downto 8);
URB40<=Input(17);
PhaseInc40<=Input(16 downto 12);
Gain40<=Input(11 downto 8);
URB44<=Input(17);
PhaseInc44<=Input(16 downto 12);
Gain44<=Input(11 downto 8);
URB48<=Input(17);
PhaseInc48<=Input(16 downto 12);
Gain48<=Input(11 downto 8);
URB52<=Input(17);
PhaseInc52<=Input(16 downto 12);
Gain52<=Input(11 downto 8);
URB56<=Input(17);
PhaseInc56<=Input(16 downto 12);
Gain56<=Input(11 downto 8);
URB60<=Input(17);
PhaseInc60<=Input(16 downto 12);
Gain60<=Input(11 downto 8);
URB64<=Input(17);
PhaseInc64<=Input(16 downto 12);
Gain64<=Input(11 downto 8);
URB68<=Input(17);
PhaseInc68<=Input(16 downto 12);
Gain68<=Input(11 downto 8);
URB72<=Input(17);
PhaseInc72<=Input(16 downto 12);
Gain72<=Input(11 downto 8);
URB76<=Input(17);
PhaseInc76<=Input(16 downto 12);
Gain76<=Input(11 downto 8);
URB80<=Input(17);
PhaseInc80<=Input(16 downto 12);
```

```
Gain80<=Input(11 downto 8);
URB84<=Input(17);
PhaseInc84<=Input(16 downto 12);
Gain84<=Input(11 downto 8);
URB88<=Input(17);
PhaseInc88<=Input(16 downto 12);
Gain88<=Input(11 downto 8);
URB92<=Input(17);
PhaseInc92<=Input(16 downto 12);
Gain92<=Input(11 downto 8);
URB96<=Input(17);
PhaseInc96<=Input(16 downto 12);
Gain96<=Input(11 downto 8);
URB100<=Input(17);
PhaseInc100<=Input(16 downto 12);
Gain100<=Input(11 downto 8);
URB104<=Input(17);
PhaseInc104<=Input(16 downto 12);
Gain104<=Input(11 downto 8);
URB108<=Input(17);
PhaseInc108<=Input(16 downto 12);
Gain108<=Input(11 downto 8);
URB112<=Input(17);
PhaseInc112<=Input(16 downto 12);
Gain112<=Input(11 downto 8);
URB116<=Input(17);
PhaseInc116<=Input(16 downto 12);
Gain116<=Input(11 downto 8);
URB120<=Input(17);
PhaseInc120<=Input(16 downto 12);
Gain120<=Input(11 downto 8);
URB124<=Input(17);
PhaseInc124<=Input(16 downto 12);
Gain124<=Input(11 downto 8);

-- Bins 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61...125 data
URB1<=Input(27);
PhaseInc1<=Input(26 downto 22);
Gain1<=Input(21 downto 18);
URB5<=Input(27);
PhaseInc5<=Input(26 downto 22);
Gain5<=Input(21 downto 18);
URB9<=Input(27);
PhaseInc9<=Input(26 downto 22);
Gain9<=Input(21 downto 18);
URB13<=Input(27);
PhaseInc13<=Input(26 downto 22);
Gain13<=Input(21 downto 18);
URB17<=Input(27);
PhaseInc17<=Input(26 downto 22);
Gain17<=Input(21 downto 18);
URB21<=Input(27);
PhaseInc21<=Input(26 downto 22);
Gain21<=Input(21 downto 18);
URB25<=Input(27);
PhaseInc25<=Input(26 downto 22);
Gain25<=Input(21 downto 18);
```

```
URB29<=Input(27);
PhaseInc29<=Input(26 downto 22);
Gain29<=Input(21 downto 18);
URB33<=Input(27);
PhaseInc33<=Input(26 downto 22);
Gain33<=Input(21 downto 18);
URB37<=Input(27);
PhaseInc37<=Input(26 downto 22);
Gain37<=Input(21 downto 18);
URB41<=Input(27);
PhaseInc41<=Input(26 downto 22);
Gain41<=Input(21 downto 18);
URB45<=Input(27);
PhaseInc45<=Input(26 downto 22);
Gain45<=Input(21 downto 18);
URB49<=Input(27);
PhaseInc49<=Input(26 downto 22);
Gain49<=Input(21 downto 18);
URB53<=Input(27);
PhaseInc53<=Input(26 downto 22);
Gain53<=Input(21 downto 18);
URB57<=Input(27);
PhaseInc57<=Input(26 downto 22);
Gain57<=Input(21 downto 18);
URB61<=Input(27);
PhaseInc61<=Input(26 downto 22);
Gain61<=Input(21 downto 18);
URB65<=Input(27);
PhaseInc65<=Input(26 downto 22);
Gain65<=Input(21 downto 18);
URB69<=Input(27);
PhaseInc69<=Input(26 downto 22);
Gain69<=Input(21 downto 18);
URB73<=Input(27);
PhaseInc73<=Input(26 downto 22);
Gain73<=Input(21 downto 18);
URB77<=Input(27);
PhaseInc77<=Input(26 downto 22);
Gain77<=Input(21 downto 18);
URB81<=Input(27);
PhaseInc81<=Input(26 downto 22);
Gain81<=Input(21 downto 18);
URB85<=Input(27);
PhaseInc85<=Input(26 downto 22);
Gain85<=Input(21 downto 18);
URB89<=Input(27);
PhaseInc89<=Input(26 downto 22);
Gain89<=Input(21 downto 18);
URB93<=Input(27);
PhaseInc93<=Input(26 downto 22);
Gain93<=Input(21 downto 18);
URB97<=Input(27);
PhaseInc97<=Input(26 downto 22);
Gain97<=Input(21 downto 18);
URB101<=Input(27);
PhaseInc101<=Input(26 downto 22);
Gain101<=Input(21 downto 18);
```

```
URB105<=Input(27);
PhaseInc105<=Input(26 downto 22);
Gain105<=Input(21 downto 18);
URB109<=Input(27);
PhaseInc109<=Input(26 downto 22);
Gain109<=Input(21 downto 18);
URB113<=Input(27);
PhaseInc113<=Input(26 downto 22);
Gain113<=Input(21 downto 18);
URB117<=Input(27);
PhaseInc117<=Input(26 downto 22);
Gain117<=Input(21 downto 18);
URB121<=Input(27);
PhaseInc121<=Input(26 downto 22);
Gain121<=Input(21 downto 18);
URB125<=Input(27);
PhaseInc125<=Input(26 downto 22);
Gain125<=Input(21 downto 18);

-- Bins 2,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62...126 data
URB2<=Input(37);
PhaseInc2<=Input(36 downto 32);
Gain2<=Input(31 downto 28);
URB6<=Input(37);
PhaseInc6<=Input(36 downto 32);
Gain6<=Input(31 downto 28);
URB10<=Input(37);
PhaseInc10<=Input(36 downto 32);
Gain10<=Input(31 downto 28);
URB14<=Input(37);
PhaseInc14<=Input(36 downto 32);
Gain14<=Input(31 downto 28);
URB18<=Input(37);
PhaseInc18<=Input(36 downto 32);
Gain18<=Input(31 downto 28);
URB22<=Input(37);
PhaseInc22<=Input(36 downto 32);
Gain22<=Input(31 downto 28);
URB26<=Input(37);
PhaseInc26<=Input(36 downto 32);
Gain26<=Input(31 downto 28);
URB30<=Input(37);
PhaseInc30<=Input(36 downto 32);
Gain30<=Input(31 downto 28);
URB34<=Input(37);
PhaseInc34<=Input(36 downto 32);
Gain34<=Input(31 downto 28);
URB38<=Input(37);
PhaseInc38<=Input(36 downto 32);
Gain38<=Input(31 downto 28);
URB42<=Input(37);
PhaseInc42<=Input(36 downto 32);
Gain42<=Input(31 downto 28);
URB46<=Input(37);
PhaseInc46<=Input(36 downto 32);
Gain46<=Input(31 downto 28);
URB50<=Input(37);
```

```
PhaseInc50<=Input(36 downto 32);
Gain50<=Input(31 downto 28);
URB54<=Input(37);
PhaseInc54<=Input(36 downto 32);
Gain54<=Input(31 downto 28);
URB58<=Input(37);
PhaseInc58<=Input(36 downto 32);
Gain58<=Input(31 downto 28);
URB62<=Input(37);
PhaseInc62<=Input(36 downto 32);
Gain62<=Input(31 downto 28);
URB66<=Input(37);
PhaseInc66<=Input(36 downto 32);
Gain66<=Input(31 downto 28);
URB70<=Input(37);
PhaseInc70<=Input(36 downto 32);
Gain70<=Input(31 downto 28);
URB74<=Input(37);
PhaseInc74<=Input(36 downto 32);
Gain74<=Input(31 downto 28);
URB78<=Input(37);
PhaseInc78<=Input(36 downto 32);
Gain78<=Input(31 downto 28);
URB82<=Input(37);
PhaseInc82<=Input(36 downto 32);
Gain82<=Input(31 downto 28);
URB86<=Input(37);
PhaseInc86<=Input(36 downto 32);
Gain86<=Input(31 downto 28);
URB90<=Input(37);
PhaseInc90<=Input(36 downto 32);
Gain90<=Input(31 downto 28);
URB94<=Input(37);
PhaseInc94<=Input(36 downto 32);
Gain94<=Input(31 downto 28);
URB98<=Input(37);
PhaseInc98<=Input(36 downto 32);
Gain98<=Input(31 downto 28);
URB102<=Input(37);
PhaseInc102<=Input(36 downto 32);
Gain102<=Input(31 downto 28);
URB106<=Input(37);
PhaseInc106<=Input(36 downto 32);
Gain106<=Input(31 downto 28);
URB110<=Input(37);
PhaseInc110<=Input(36 downto 32);
Gain110<=Input(31 downto 28);
URB114<=Input(37);
PhaseInc114<=Input(36 downto 32);
Gain114<=Input(31 downto 28);
URB118<=Input(37);
PhaseInc118<=Input(36 downto 32);
Gain118<=Input(31 downto 28);
URB122<=Input(37);
PhaseInc122<=Input(36 downto 32);
Gain122<=Input(31 downto 28);
URB126<=Input(37);
```

```
PhaseInc126<=Input(36 downto 32);
Gain126<=Input(31 downto 28);

-- Bins 3,7,11,15,19,23,27,31,35,39,43,47,51,55,59,63...127 data
URB3<=Input(47);
PhaseInc3<=Input(46 downto 42);
Gain3<=Input(41 downto 38);
URB7<=Input(47);
PhaseInc7<=Input(46 downto 42);
Gain7<=Input(41 downto 38);
URB11<=Input(47);
PhaseInc11<=Input(46 downto 42);
Gain11<=Input(41 downto 38);
URB15<=Input(47);
PhaseInc15<=Input(46 downto 42);
Gain15<=Input(41 downto 38);
URB19<=Input(47);
PhaseInc19<=Input(46 downto 42);
Gain19<=Input(41 downto 38);
URB23<=Input(47);
PhaseInc23<=Input(46 downto 42);
Gain23<=Input(41 downto 38);
URB27<=Input(47);
PhaseInc27<=Input(46 downto 42);
Gain27<=Input(41 downto 38);
URB31<=Input(47);
PhaseInc31<=Input(46 downto 42);
Gain31<=Input(41 downto 38);
URB35<=Input(47);
PhaseInc35<=Input(46 downto 42);
Gain35<=Input(41 downto 38);
URB39<=Input(47);
PhaseInc39<=Input(46 downto 42);
Gain39<=Input(41 downto 38);
URB43<=Input(47);
PhaseInc43<=Input(46 downto 42);
Gain43<=Input(41 downto 38);
URB47<=Input(47);
PhaseInc47<=Input(46 downto 42);
Gain47<=Input(41 downto 38);
URB51<=Input(47);
PhaseInc51<=Input(46 downto 42);
Gain51<=Input(41 downto 38);
URB55<=Input(47);
PhaseInc55<=Input(46 downto 42);
Gain55<=Input(41 downto 38);
URB59<=Input(47);
PhaseInc59<=Input(46 downto 42);
Gain59<=Input(41 downto 38);
URB63<=Input(47);
PhaseInc63<=Input(46 downto 42);
Gain63<=Input(41 downto 38);
URB67<=Input(47);
PhaseInc67<=Input(46 downto 42);
Gain67<=Input(41 downto 38);
URB71<=Input(47);
PhaseInc71<=Input(46 downto 42);
```

```
Gain71<=Input(41 downto 38);
URB75<=Input(47);
PhaseInc75<=Input(46 downto 42);
Gain75<=Input(41 downto 38);
URB79<=Input(47);
PhaseInc79<=Input(46 downto 42);
Gain79<=Input(41 downto 38);
URB83<=Input(47);
PhaseInc83<=Input(46 downto 42);
Gain83<=Input(41 downto 38);
URB87<=Input(47);
PhaseInc87<=Input(46 downto 42);
Gain87<=Input(41 downto 38);
URB91<=Input(47);
PhaseInc91<=Input(46 downto 42);
Gain91<=Input(41 downto 38);
URB95<=Input(47);
PhaseInc95<=Input(46 downto 42);
Gain95<=Input(41 downto 38);
URB99<=Input(47);
PhaseInc99<=Input(46 downto 42);
Gain99<=Input(41 downto 38);
URB103<=Input(47);
PhaseInc103<=Input(46 downto 42);
Gain103<=Input(41 downto 38);
URB107<=Input(47);
PhaseInc107<=Input(46 downto 42);
Gain107<=Input(41 downto 38);
URB111<=Input(47);
PhaseInc111<=Input(46 downto 42);
Gain111<=Input(41 downto 38);
URB115<=Input(47);
PhaseInc115<=Input(46 downto 42);
Gain115<=Input(41 downto 38);
URB119<=Input(47);
PhaseInc119<=Input(46 downto 42);
Gain119<=Input(41 downto 38);
URB123<=Input(47);
PhaseInc123<=Input(46 downto 42);
Gain123<=Input(41 downto 38);
URB127<=Input(47);
PhaseInc127<=Input(46 downto 42);
Gain127<=Input(41 downto 38);

DFF0: DFlipFlop port map (CLK, LD, RESET, PRB, PRB1, PRB1not);
DFF1: DFlipFlop port map (CLK, LD, RESET, PRB1, PRB2, PRB2not);
DFF2: DFlipFlop port map (CLK, LD, RESET, PRB2, PRB3, PRB3not);
DFF3: DFlipFlop port map (CLK, LD, RESET, PRB3, PRB4, PRB4not);
DFF4: DFlipFlop port map (CLK, LD, RESET, PRB4, PRB5, PRB5not);
DFF5: DFlipFlop port map (CLK, LD, RESET, PRB5, PRB6, PRB6not);
DFF6: DFlipFlop port map (CLK, LD, RESET, PRB6, PRB7, PRB7not);
DFF7: DFlipFlop port map (CLK, LD, RESET, PRB7, PRB8, PRB8not);
DFF8: DFlipFlop port map (CLK, LD, RESET, PRB8, PRB9, PRB9not);
DFF9: DFlipFlop port map (CLK, LD, RESET, PRB9, PRB10, PRB10not);
DFF10: DFlipFlop port map (CLK, LD, RESET, PRB10, PRB11, PRB11not);
DFF11: DFlipFlop port map (CLK, LD, RESET, PRB11, PRB12, PRB12not);
DFF12: DFlipFlop port map (CLK, LD, RESET, PRB12, PRB13, PRB13not);
```

```
DFF13: DFlipFlop port map (CLK, LD, RESET, PRB13, PRB14, PRB14not);
DFF14: DFlipFlop port map (CLK, LD, RESET, PRB14, PRB15, PRB15not);
DFF15: DFlipFlop port map (CLK, LD, RESET, PRB15, PRB16, PRB16not);
DFF16: DFlipFlop port map (CLK, LD, RESET, PRB16, PRB17, PRB17not);
DFF17: DFlipFlop port map (CLK, LD, RESET, PRB17, PRB18, PRB18not);
DFF18: DFlipFlop port map (CLK, LD, RESET, PRB18, PRB19, PRB19not);
DFF19: DFlipFlop port map (CLK, LD, RESET, PRB19, PRB20, PRB20not);
DFF20: DFlipFlop port map (CLK, LD, RESET, PRB20, PRB21, PRB21not);
DFF21: DFlipFlop port map (CLK, LD, RESET, PRB21, PRB22, PRB22not);
DFF22: DFlipFlop port map (CLK, LD, RESET, PRB22, PRB23, PRB23not);
DFF23: DFlipFlop port map (CLK, LD, RESET, PRB23, PRB24, PRB24not);
DFF24: DFlipFlop port map (CLK, LD, RESET, PRB24, PRB25, PRB25not);
DFF25: DFlipFlop port map (CLK, LD, RESET, PRB25, PRB26, PRB26not);
DFF26: DFlipFlop port map (CLK, LD, RESET, PRB26, PRB27, PRB27not);
DFF27: DFlipFlop port map (CLK, LD, RESET, PRB27, PRB28, PRB28not);
DFF28: DFlipFlop port map (CLK, LD, RESET, PRB28, PRB29, PRB29not);
DFF29: DFlipFlop port map (CLK, LD, RESET, PRB29, PRB30, PRB30not);
DFF30: DFlipFlop port map (CLK, LD, RESET, PRB30, PRB31, PRB31not);

BIN0: OneBin port map (DRFM1, PhaseInc0, Gain0, URB0, UNP, PRB,
ODVout1, PSVout1, Q1, I1, Q, I, ODVout0, PSVout0, DRFM0, CLK);
BIN1: OneBin port map (DRFM2, PhaseInc1, Gain1, URB1, UNP, PRB,
ODVout2, PSVout2, Q2, I2, Q1, I1, ODVout1, PSVout1, DRFM1, CLK);
BIN2: OneBin port map (DRFM3, PhaseInc2, Gain2, URB2, UNP, PRB,
ODVout3, PSVout3, Q3, I3, Q2, I2, ODVout2, PSVout2, DRFM2, CLK);
BIN3: OneBin port map (DRFM4, PhaseInc3, Gain3, URB3, UNP, PRB,
ODVout4, PSVout4, Q4, I4, Q3, I3, ODVout3, PSVout3, DRFM3, CLK);
BIN4: OneBin port map (DRFM5, PhaseInc4, Gain4, URB4, UNP, PRB1,
ODVout5, PSVout5, Q5, I5, Q4, I4, ODVout4, PSVout4, DRFM4, CLK);
BIN5: OneBin port map (DRFM6, PhaseInc5, Gain5, URB5, UNP, PRB1,
ODVout6, PSVout6, Q6, I6, Q5, I5, ODVout5, PSVout5, DRFM5, CLK);
BIN6: OneBin port map (DRFM7, PhaseInc6, Gain6, URB6, UNP, PRB1,
ODVout7, PSVout7, Q7, I7, Q6, I6, ODVout6, PSVout6, DRFM6, CLK);
BIN7: OneBin port map (DRFM8, PhaseInc7, Gain7, URB7, UNP, PRB1,
ODVout8, PSVout8, Q8, I8, Q7, I7, ODVout7, PSVout7, DRFM7, CLK);
BIN8: OneBin port map (DRFM9, PhaseInc8, Gain8, URB8, UNP, PRB2,
ODVout9, PSVout9, Q9, I9, Q8, I8, ODVout8, PSVout8, DRFM8, CLK);
BIN9: OneBin port map (DRFM10, PhaseInc9, Gain9, URB9, UNP, PRB2,
ODVout10, PSVout10, Q10, I10, Q9, I9, ODVout9, PSVout9, DRFM9, CLK);
BIN10: OneBin port map (DRFM11, PhaseInc10, Gain10, URB10, UNP, PRB2,
ODVout11, PSVout11, Q11, I11, Q10, I10, ODVout10, PSVout10, DRFM10,
CLK);
BIN11: OneBin port map (DRFM12, PhaseInc11, Gain11, URB11, UNP, PRB2,
ODVout12, PSVout12, Q12, I12, Q11, I11, ODVout11, PSVout11, DRFM11,
CLK);
BIN12: OneBin port map (DRFM13, PhaseInc12, Gain12, URB12, UNP, PRB3,
ODVout13, PSVout13, Q13, I13, Q12, I12, ODVout12, PSVout12, DRFM12,
CLK);
BIN13: OneBin port map (DRFM14, PhaseInc13, Gain13, URB13, UNP, PRB3,
ODVout14, PSVout14, Q14, I14, Q13, I13, ODVout13, PSVout13, DRFM13,
CLK);
BIN14: OneBin port map (DRFM15, PhaseInc14, Gain14, URB14, UNP, PRB3,
ODVout15, PSVout15, Q15, I15, Q14, I14, ODVout14, PSVout14, DRFM14,
CLK);
BIN15: OneBin port map (DRFM16, PhaseInc15, Gain15, URB15, UNP, PRB3,
ODVout16, PSVout16, Q16, I16, Q15, I15, ODVout15, PSVout15, DRFM15,
CLK);
```

```
BIN16: OneBin port map (DRFM17, PhaseInc16, Gain16, URB16, UNP, PRB4,
ODVout17, PSVout17, Q17, I17, Q16, I16, ODVout16, PSVout16, DRFM16,
CLK);
BIN17: OneBin port map (DRFM18, PhaseInc17, Gain17, URB17, UNP, PRB4,
ODVout18, PSVout18, Q18, I18, Q17, I17, ODVout17, PSVout17, DRFM17,
CLK);
BIN18: OneBin port map (DRFM19, PhaseInc18, Gain18, URB18, UNP, PRB4,
ODVout19, PSVout19, Q19, I19, Q18, I18, ODVout18, PSVout18, DRFM18,
CLK);
BIN19: OneBin port map (DRFM20, PhaseInc19, Gain19, URB19, UNP, PRB4,
ODVout20, PSVout20, Q20, I20, Q19, I19, ODVout19, PSVout19, DRFM19,
CLK);
BIN20: OneBin port map (DRFM21, PhaseInc20, Gain20, URB20, UNP, PRB5,
ODVout21, PSVout21, Q21, I21, Q20, I20, ODVout20, PSVout20, DRFM20,
CLK);
BIN21: OneBin port map (DRFM22, PhaseInc21, Gain21, URB21, UNP, PRB5,
ODVout22, PSVout22, Q22, I22, Q21, I21, ODVout21, PSVout21, DRFM21,
CLK);
BIN22: OneBin port map (DRFM23, PhaseInc22, Gain22, URB22, UNP, PRB5,
ODVout23, PSVout23, Q23, I23, Q22, I22, ODVout22, PSVout22, DRFM22,
CLK);
BIN23: OneBin port map (DRFM24, PhaseInc23, Gain23, URB23, UNP, PRB5,
ODVout24, PSVout24, Q24, I24, Q23, I23, ODVout23, PSVout23, DRFM23,
CLK);
BIN24: OneBin port map (DRFM25, PhaseInc24, Gain24, URB24, UNP, PRB6,
ODVout25, PSVout25, Q25, I25, Q24, I24, ODVout24, PSVout24, DRFM24,
CLK);
BIN25: OneBin port map (DRFM26, PhaseInc25, Gain25, URB25, UNP, PRB6,
ODVout26, PSVout26, Q26, I26, Q25, I25, ODVout25, PSVout25, DRFM25,
CLK);
BIN26: OneBin port map (DRFM27, PhaseInc26, Gain26, URB26, UNP, PRB6,
ODVout27, PSVout27, Q27, I27, Q26, I26, ODVout26, PSVout26, DRFM26,
CLK);
BIN27: OneBin port map (DRFM28, PhaseInc27, Gain27, URB27, UNP, PRB6,
ODVout28, PSVout28, Q28, I28, Q27, I27, ODVout27, PSVout27, DRFM27,
CLK);
BIN28: OneBin port map (DRFM29, PhaseInc28, Gain28, URB28, UNP, PRB7,
ODVout29, PSVout29, Q29, I29, Q28, I28, ODVout28, PSVout28, DRFM28,
CLK);
BIN29: OneBin port map (DRFM30, PhaseInc29, Gain29, URB29, UNP, PRB7,
ODVout30, PSVout30, Q30, I30, Q29, I29, ODVout29, PSVout29, DRFM29,
CLK);
BIN30: OneBin port map (DRFM31, PhaseInc30, Gain30, URB30, UNP, PRB7,
ODVout31, PSVout31, Q31, I31, Q30, I30, ODVout30, PSVout30, DRFM30,
CLK);
BIN31: OneBin port map (DRFM32, PhaseInc31, Gain31, URB31, UNP, PRB7,
ODVout32, PSVout32, Q32, I32, Q31, I31, ODVout31, PSVout31, DRFM31,
CLK);
BIN32: OneBin port map (DRFM33, PhaseInc32, Gain32, URB32, UNP, PRB8,
ODVout33, PSVout33, Q33, I33, Q32, I32, ODVout32, PSVout32, DRFM32,
CLK);
BIN33: OneBin port map (DRFM34, PhaseInc33, Gain33, URB33, UNP, PRB8,
ODVout34, PSVout34, Q34, I34, Q33, I33, ODVout33, PSVout33, DRFM33,
CLK);
BIN34: OneBin port map (DRFM35, PhaseInc34, Gain34, URB34, UNP, PRB8,
ODVout35, PSVout35, Q35, I35, Q34, I34, ODVout34, PSVout34, DRFM34,
CLK);
```

```
BIN35: OneBin port map (DRFM36, PhaseInc35, Gain35, URB35, UNP, PRB8,
ODVout36, PSVout36, Q36, I36, Q35, I35, ODVout35, PSVout35, DRFM35,
CLK);
BIN36: OneBin port map (DRFM37, PhaseInc36, Gain36, URB36, UNP, PRB9,
ODVout37, PSVout37, Q37, I37, Q36, I36, ODVout36, PSVout36, DRFM36,
CLK);
BIN37: OneBin port map (DRFM38, PhaseInc37, Gain37, URB37, UNP, PRB9,
ODVout38, PSVout38, Q38, I38, Q37, I37, ODVout37, PSVout37, DRFM37,
CLK);
BIN38: OneBin port map (DRFM39, PhaseInc38, Gain38, URB38, UNP, PRB9,
ODVout39, PSVout39, Q39, I39, Q38, I38, ODVout38, PSVout38, DRFM38,
CLK);
BIN39: OneBin port map (DRFM40, PhaseInc39, Gain39, URB39, UNP, PRB9,
ODVout40, PSVout40, Q40, I40, Q39, I39, ODVout39, PSVout39, DRFM39,
CLK);
BIN40: OneBin port map (DRFM41, PhaseInc40, Gain40, URB40, UNP, PRB10,
ODVout41, PSVout41, Q41, I41, Q40, I40, ODVout40, PSVout40, DRFM40,
CLK);
BIN41: OneBin port map (DRFM42, PhaseInc41, Gain41, URB41, UNP, PRB10,
ODVout42, PSVout42, Q42, I42, Q41, I41, ODVout41, PSVout41, DRFM41,
CLK);
BIN42: OneBin port map (DRFM43, PhaseInc42, Gain42, URB42, UNP, PRB10,
ODVout43, PSVout43, Q43, I43, Q42, I42, ODVout42, PSVout42, DRFM42,
CLK);
BIN43: OneBin port map (DRFM44, PhaseInc43, Gain43, URB43, UNP, PRB10,
ODVout44, PSVout44, Q44, I44, Q43, I43, ODVout43, PSVout43, DRFM43,
CLK);
BIN44: OneBin port map (DRFM45, PhaseInc44, Gain44, URB44, UNP, PRB11,
ODVout45, PSVout45, Q45, I45, Q44, I44, ODVout44, PSVout44, DRFM44,
CLK);
BIN45: OneBin port map (DRFM46, PhaseInc45, Gain45, URB45, UNP, PRB11,
ODVout46, PSVout46, Q46, I46, Q45, I45, ODVout45, PSVout45, DRFM45,
CLK);
BIN46: OneBin port map (DRFM47, PhaseInc46, Gain46, URB46, UNP, PRB11,
ODVout47, PSVout47, Q47, I47, Q46, I46, ODVout46, PSVout46, DRFM46,
CLK);
BIN47: OneBin port map (DRFM48, PhaseInc47, Gain47, URB47, UNP, PRB11,
ODVout48, PSVout48, Q48, I48, Q47, I47, ODVout47, PSVout47, DRFM47,
CLK);
BIN48: OneBin port map (DRFM49, PhaseInc48, Gain48, URB48, UNP, PRB12,
ODVout49, PSVout49, Q49, I49, Q48, I48, ODVout48, PSVout48, DRFM48,
CLK);
BIN49: OneBin port map (DRFM50, PhaseInc49, Gain49, URB49, UNP, PRB12,
ODVout50, PSVout50, Q50, I50, Q49, I49, ODVout49, PSVout49, DRFM49,
CLK);
BIN50: OneBin port map (DRFM51, PhaseInc50, Gain50, URB50, UNP, PRB12,
ODVout51, PSVout51, Q51, I51, Q50, I50, ODVout50, PSVout50, DRFM50,
CLK);
BIN51: OneBin port map (DRFM52, PhaseInc51, Gain51, URB51, UNP, PRB12,
ODVout52, PSVout52, Q52, I52, Q51, I51, ODVout51, PSVout51, DRFM51,
CLK);
BIN52: OneBin port map (DRFM53, PhaseInc52, Gain52, URB52, UNP, PRB13,
ODVout53, PSVout53, Q53, I53, Q52, I52, ODVout52, PSVout52, DRFM52,
CLK);
BIN53: OneBin port map (DRFM54, PhaseInc53, Gain53, URB53, UNP, PRB13,
ODVout54, PSVout54, Q54, I54, Q53, I53, ODVout53, PSVout53, DRFM53,
CLK);
```

```
BIN54: OneBin port map (DRFM55, PhaseInc54, Gain54, URB54, UNP, PRB13,
ODVout55, PSVout55, Q55, I55, Q54, I54, ODVout54, PSVout54, DRFM54,
CLK);
BIN55: OneBin port map (DRFM56, PhaseInc55, Gain55, URB55, UNP, PRB13,
ODVout56, PSVout56, Q56, I56, Q55, I55, ODVout55, PSVout55, DRFM55,
CLK);
BIN56: OneBin port map (DRFM57, PhaseInc56, Gain56, URB56, UNP, PRB14,
ODVout57, PSVout57, Q57, I57, Q56, I56, ODVout56, PSVout56, DRFM56,
CLK);
BIN57: OneBin port map (DRFM58, PhaseInc57, Gain57, URB57, UNP, PRB14,
ODVout58, PSVout58, Q58, I58, Q57, I57, ODVout57, PSVout57, DRFM57,
CLK);
BIN58: OneBin port map (DRFM59, PhaseInc58, Gain58, URB58, UNP, PRB14,
ODVout59, PSVout59, Q59, I59, Q58, I58, ODVout58, PSVout58, DRFM58,
CLK);
BIN59: OneBin port map (DRFM60, PhaseInc59, Gain59, URB59, UNP, PRB14,
ODVout60, PSVout60, Q60, I60, Q59, I59, ODVout59, PSVout59, DRFM59,
CLK);
BIN60: OneBin port map (DRFM61, PhaseInc60, Gain60, URB60, UNP, PRB15,
ODVout61, PSVout61, Q61, I61, Q60, I60, ODVout60, PSVout60, DRFM60,
CLK);
BIN61: OneBin port map (DRFM62, PhaseInc61, Gain61, URB61, UNP, PRB15,
ODVout62, PSVout62, Q62, I62, Q61, I61, ODVout61, PSVout61, DRFM61,
CLK);
BIN62: OneBin port map (DRFM63, PhaseInc62, Gain62, URB62, UNP, PRB15,
ODVout63, PSVout63, Q63, I63, Q62, I62, ODVout62, PSVout62, DRFM62,
CLK);
BIN63: OneBin port map (DRFM64, PhaseInc63, Gain63, URB63, UNP, PRB15,
ODVout64, PSVout64, Q64, I64, Q63, I63, ODVout63, PSVout63, DRFM63,
CLK);
BIN64: OneBin port map (DRFM65, PhaseInc64, Gain64, URB64, UNP, PRB16,
ODVout65, PSVout65, Q65, I65, Q64, I64, ODVout64, PSVout64, DRFM64,
CLK);
BIN65: OneBin port map (DRFM66, PhaseInc65, Gain65, URB65, UNP, PRB16,
ODVout66, PSVout66, Q66, I66, Q65, I65, ODVout65, PSVout65, DRFM65,
CLK);
BIN66: OneBin port map (DRFM67, PhaseInc66, Gain66, URB66, UNP, PRB16,
ODVout67, PSVout67, Q67, I67, Q66, I66, ODVout66, PSVout66, DRFM66,
CLK);
BIN67: OneBin port map (DRFM68, PhaseInc67, Gain67, URB67, UNP, PRB16,
ODVout68, PSVout68, Q68, I68, Q67, I67, ODVout67, PSVout67, DRFM67,
CLK);
BIN68: OneBin port map (DRFM69, PhaseInc68, Gain68, URB68, UNP, PRB17,
ODVout69, PSVout69, Q69, I69, Q68, I68, ODVout68, PSVout68, DRFM68,
CLK);
BIN69: OneBin port map (DRFM70, PhaseInc69, Gain69, URB69, UNP, PRB17,
ODVout70, PSVout70, Q70, I70, Q69, I69, ODVout69, PSVout69, DRFM69,
CLK);
BIN70: OneBin port map (DRFM71, PhaseInc70, Gain70, URB70, UNP, PRB17,
ODVout71, PSVout71, Q71, I71, Q70, I70, ODVout70, PSVout70, DRFM70,
CLK);
BIN71: OneBin port map (DRFM72, PhaseInc71, Gain71, URB71, UNP, PRB17,
ODVout72, PSVout72, Q72, I72, Q71, I71, ODVout71, PSVout71, DRFM71,
CLK);
BIN72: OneBin port map (DRFM73, PhaseInc72, Gain72, URB72, UNP, PRB18,
ODVout73, PSVout73, Q73, I73, Q72, I72, ODVout72, PSVout72, DRFM72,
CLK);
```

```
BIN73: OneBin port map (DRFM74, PhaseInc73, Gain73, URB73, UNP, PRB18,
ODVout74, PSVout74, Q74, I74, Q73, I73, ODVout73, PSVout73, DRFM73,
CLK);
BIN74: OneBin port map (DRFM75, PhaseInc74, Gain74, URB74, UNP, PRB18,
ODVout75, PSVout75, Q75, I75, Q74, I74, ODVout74, PSVout74, DRFM74,
CLK);
BIN75: OneBin port map (DRFM76, PhaseInc75, Gain75, URB75, UNP, PRB18,
ODVout76, PSVout76, Q76, I76, Q75, I75, ODVout75, PSVout75, DRFM75,
CLK);
BIN76: OneBin port map (DRFM77, PhaseInc76, Gain76, URB76, UNP, PRB19,
ODVout77, PSVout77, Q77, I77, Q76, I76, ODVout76, PSVout76, DRFM76,
CLK);
BIN77: OneBin port map (DRFM78, PhaseInc77, Gain77, URB77, UNP, PRB19,
ODVout78, PSVout78, Q78, I78, Q77, I77, ODVout77, PSVout77, DRFM77,
CLK);
BIN78: OneBin port map (DRFM79, PhaseInc78, Gain78, URB78, UNP, PRB19,
ODVout79, PSVout79, Q79, I79, Q78, I78, ODVout78, PSVout78, DRFM78,
CLK);
BIN79: OneBin port map (DRFM80, PhaseInc79, Gain79, URB79, UNP, PRB19,
ODVout80, PSVout80, Q80, I80, Q79, I79, ODVout79, PSVout79, DRFM79,
CLK);
BIN80: OneBin port map (DRFM81, PhaseInc80, Gain80, URB80, UNP, PRB20,
ODVout81, PSVout81, Q81, I81, Q80, I80, ODVout80, PSVout80, DRFM80,
CLK);
BIN81: OneBin port map (DRFM82, PhaseInc81, Gain81, URB81, UNP, PRB20,
ODVout82, PSVout82, Q82, I82, Q81, I81, ODVout81, PSVout81, DRFM81,
CLK);
BIN82: OneBin port map (DRFM83, PhaseInc82, Gain82, URB82, UNP, PRB20,
ODVout83, PSVout83, Q83, I83, Q82, I82, ODVout82, PSVout82, DRFM82,
CLK);
BIN83: OneBin port map (DRFM84, PhaseInc83, Gain83, URB83, UNP, PRB20,
ODVout84, PSVout84, Q84, I84, Q83, I83, ODVout83, PSVout83, DRFM83,
CLK);
BIN84: OneBin port map (DRFM85, PhaseInc84, Gain84, URB84, UNP, PRB21,
ODVout85, PSVout85, Q85, I85, Q84, I84, ODVout84, PSVout84, DRFM84,
CLK);
BIN85: OneBin port map (DRFM86, PhaseInc85, Gain85, URB85, UNP, PRB21,
ODVout86, PSVout86, Q86, I86, Q85, I85, ODVout85, PSVout85, DRFM85,
CLK);
BIN86: OneBin port map (DRFM87, PhaseInc86, Gain86, URB86, UNP, PRB21,
ODVout87, PSVout87, Q87, I87, Q86, I86, ODVout86, PSVout86, DRFM86,
CLK);
BIN87: OneBin port map (DRFM88, PhaseInc87, Gain87, URB87, UNP, PRB21,
ODVout88, PSVout88, Q88, I88, Q87, I87, ODVout87, PSVout87, DRFM87,
CLK);
BIN88: OneBin port map (DRFM89, PhaseInc88, Gain88, URB88, UNP, PRB22,
ODVout89, PSVout89, Q89, I89, Q88, I88, ODVout88, PSVout88, DRFM88,
CLK);
BIN89: OneBin port map (DRFM90, PhaseInc89, Gain89, URB89, UNP, PRB22,
ODVout90, PSVout90, Q90, I90, Q89, I89, ODVout89, PSVout89, DRFM89,
CLK);
BIN90: OneBin port map (DRFM91, PhaseInc90, Gain90, URB90, UNP, PRB22,
ODVout91, PSVout91, Q91, I91, Q90, I90, ODVout90, PSVout90, DRFM90,
CLK);
BIN91: OneBin port map (DRFM92, PhaseInc91, Gain91, URB91, UNP, PRB22,
ODVout92, PSVout92, Q92, I92, Q91, I91, ODVout91, PSVout91, DRFM91,
CLK);
```

```
BIN92: OneBin port map (DRFM93, PhaseInc92, Gain92, URB92, UNP, PRB23,
ODVout93, PSVout93, Q93, I93, Q92, I92, ODVout92, PSVout92, DRFM92,
CLK);
BIN93: OneBin port map (DRFM94, PhaseInc93, Gain93, URB93, UNP, PRB23,
ODVout94, PSVout94, Q94, I94, Q93, I93, ODVout93, PSVout93, DRFM93,
CLK);
BIN94: OneBin port map (DRFM95, PhaseInc94, Gain94, URB94, UNP, PRB23,
ODVout95, PSVout95, Q95, I95, Q94, I94, ODVout94, PSVout94, DRFM94,
CLK);
BIN95: OneBin port map (DRFM96, PhaseInc95, Gain95, URB95, UNP, PRB23,
ODVout96, PSVout96, Q96, I96, Q95, I95, ODVout95, PSVout95, DRFM95,
CLK);
BIN96: OneBin port map (DRFM97, PhaseInc96, Gain96, URB96, UNP, PRB24,
ODVout97, PSVout97, Q97, I97, Q96, I96, ODVout96, PSVout96, DRFM96,
CLK);
BIN97: OneBin port map (DRFM98, PhaseInc97, Gain97, URB97, UNP, PRB24,
ODVout98, PSVout98, Q98, I98, Q97, I97, ODVout97, PSVout97, DRFM97,
CLK);
BIN98: OneBin port map (DRFM99, PhaseInc98, Gain98, URB98, UNP, PRB24,
ODVout99, PSVout99, Q99, I99, Q98, I98, ODVout98, PSVout98, DRFM98,
CLK);
BIN99: OneBin port map (DRFM100, PhaseInc99, Gain99, URB99, UNP, PRB24,
ODVout100, PSVout100, Q100, I100, Q99, I99, ODVout99, PSVout99, DRFM99,
CLK);
BIN100: OneBin port map (DRFM101, PhaseInc100, Gain100, URB100, UNP,
PRB25, ODVout101, PSVout101, Q101, I101, Q100, I100, ODVout100,
PSVout100, DRFM100, CLK);
BIN101: OneBin port map (DRFM102, PhaseInc101, Gain101, URB101, UNP,
PRB25, ODVout102, PSVout102, Q102, I102, Q101, I101, ODVout101,
PSVout101, DRFM101, CLK);
BIN102: OneBin port map (DRFM103, PhaseInc102, Gain102, URB102, UNP,
PRB25, ODVout103, PSVout103, Q103, I103, Q102, I102, ODVout102,
PSVout102, DRFM102, CLK);
BIN103: OneBin port map (DRFM104, PhaseInc103, Gain103, URB103, UNP,
PRB25, ODVout104, PSVout104, Q104, I104, Q103, I103, ODVout103,
PSVout103, DRFM103, CLK);
BIN104: OneBin port map (DRFM105, PhaseInc104, Gain104, URB104, UNP,
PRB26, ODVout105, PSVout105, Q105, I105, Q104, I104, ODVout104,
PSVout104, DRFM104, CLK);
BIN105: OneBin port map (DRFM106, PhaseInc105, Gain105, URB105, UNP,
PRB26, ODVout106, PSVout106, Q106, I106, Q105, I105, ODVout105,
PSVout105, DRFM105, CLK);
BIN106: OneBin port map (DRFM107, PhaseInc106, Gain106, URB106, UNP,
PRB26, ODVout107, PSVout107, Q107, I107, Q106, I106, ODVout106,
PSVout106, DRFM106, CLK);
BIN107: OneBin port map (DRFM108, PhaseInc107, Gain107, URB107, UNP,
PRB26, ODVout108, PSVout108, Q108, I108, Q107, I107, ODVout107,
PSVout107, DRFM107, CLK);
BIN108: OneBin port map (DRFM109, PhaseInc108, Gain108, URB108, UNP,
PRB27, ODVout109, PSVout109, Q109, I109, Q108, I108, ODVout108,
PSVout108, DRFM108, CLK);
BIN109: OneBin port map (DRFM110, PhaseInc109, Gain109, URB109, UNP,
PRB27, ODVout110, PSVout110, Q110, I110, Q109, I109, ODVout109,
PSVout109, DRFM109, CLK);
BIN110: OneBin port map (DRFM111, PhaseInc110, Gain110, URB110, UNP,
PRB27, ODVout111, PSVout111, Q111, I111, Q110, I110, ODVout110,
PSVout110, DRFM110, CLK);
```

```
BIN111: OneBin port map (DRFM112, PhaseInc111, Gain111, URB111, UNP,
PRB27, ODVout112, PSVout112, Q112, I112, Q111, I111, ODVout111,
PSVout111, DRFM111, CLK);
BIN112: OneBin port map (DRFM113, PhaseInc112, Gain112, URB112, UNP,
PRB28, ODVout113, PSVout113, Q113, I113, Q112, I112, ODVout112,
PSVout112, DRFM112, CLK);
BIN113: OneBin port map (DRFM114, PhaseInc113, Gain113, URB113, UNP,
PRB28, ODVout114, PSVout114, Q114, I114, Q113, I113, ODVout113,
PSVout113, DRFM113, CLK);
BIN114: OneBin port map (DRFM115, PhaseInc114, Gain114, URB114, UNP,
PRB28, ODVout115, PSVout115, Q115, I115, Q114, I114, ODVout114,
PSVout114, DRFM114, CLK);
BIN115: OneBin port map (DRFM116, PhaseInc115, Gain115, URB115, UNP,
PRB28, ODVout116, PSVout116, Q116, I116, Q115, I115, ODVout115,
PSVout115, DRFM115, CLK);
BIN116: OneBin port map (DRFM117, PhaseInc116, Gain116, URB116, UNP,
PRB29, ODVout117, PSVout117, Q117, I117, Q116, I116, ODVout116,
PSVout116, DRFM116, CLK);
BIN117: OneBin port map (DRFM118, PhaseInc117, Gain117, URB117, UNP,
PRB29, ODVout118, PSVout118, Q118, I118, Q117, I117, ODVout117,
PSVout117, DRFM117, CLK);
BIN118: OneBin port map (DRFM119, PhaseInc118, Gain118, URB118, UNP,
PRB29, ODVout119, PSVout119, Q119, I119, Q118, I118, ODVout118,
PSVout118, DRFM118, CLK);
BIN119: OneBin port map (DRFM120, PhaseInc119, Gain119, URB119, UNP,
PRB29, ODVout120, PSVout120, Q120, I120, Q119, I119, ODVout119,
PSVout119, DRFM119, CLK);
BIN120: OneBin port map (DRFM121, PhaseInc120, Gain120, URB120, UNP,
PRB30, ODVout121, PSVout121, Q121, I121, Q120, I120, ODVout120,
PSVout120, DRFM120, CLK);
BIN121: OneBin port map (DRFM122, PhaseInc121, Gain121, URB121, UNP,
PRB30, ODVout122, PSVout122, Q122, I122, Q121, I121, ODVout121,
PSVout121, DRFM121, CLK);
BIN122: OneBin port map (DRFM123, PhaseInc122, Gain122, URB122, UNP,
PRB30, ODVout123, PSVout123, Q123, I123, Q122, I122, ODVout122,
PSVout122, DRFM122, CLK);
BIN123: OneBin port map (DRFM124, PhaseInc123, Gain123, URB123, UNP,
PRB30, ODVout124, PSVout124, Q124, I124, Q123, I123, ODVout123,
PSVout123, DRFM123, CLK);
BIN124: OneBin port map (DRFM125, PhaseInc124, Gain124, URB124, UNP,
PRB31, ODVout125, PSVout125, Q125, I125, Q124, I124, ODVout124,
PSVout124, DRFM124, CLK);
BIN125: OneBin port map (DRFM126, PhaseInc125, Gain125, URB125, UNP,
PRB31, ODVout126, PSVout126, Q126, I126, Q125, I125, ODVout125,
PSVout125, DRFM125, CLK);
BIN126: OneBin port map (DRFM127, PhaseInc126, Gain126, URB126, UNP,
PRB31, ODVout127, PSVout127, Q127, I127, Q126, I126, ODVout126,
PSVout126, DRFM126, CLK);
BIN127: OneBin port map (PhaseSamp, PhaseInc127, Gain127, URB127, UNP,
PRB31, ODVin, PSVin, "00000000000000000", "00000000000000000", Q127,
I127,ODVout127, PSVout127, DRFM127, CLK);
-- Output Format:
-- 63-41 |   40   |   39   | 38-22 | 21-5 | 4-0
--unused | PSVout | ODVout |   Q   |   I  | DRFM

Output(40)<=PSVout0;
Output(39)<=ODVout0;
Output(38 downto 22)<=Q;
Output(21 downto 5)<=I;
Output(4 downto 0)<=DRFM0;
Output(63 downto 41)<="00000000000000000000000";

end One28Bin;
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B

This appendix contains the support files required to compile and execute the VHDL macros on the SRC-6E.

A. 4 BIN .BOX FILE

```
module FourBin (Input, Output, CLK) /* synthesis syn_black_box */ ;
      input [63:0] Input;
      output [63:0] Output;
      input CLK;
endmodule
```

B. 4 BIN .INFO FILE

```
BEGIN_DEF "Four_Bin"
      MACRO = "FourBin";
      STATEFUL = NO;
      EXTERNAL = NO;
      PIPELINED = YES;
      LATENCY = 21;

      INPUTS = 1:
      I0 = INT 64 BITS (Input[63:0])
      ;

      OUTPUTS = 1:
      O0 = INT 64 BITS (Output[63:0])
      ;

      IN_SIGNAL : 1 BITS "CLK"="CLOCK";

END_DEF
```

C. 4 BIN .MC FILE

```
/* FourBinS.mc */
#include <libmap.h>

#define IBANK MAX_OBM_SIZE

void FourBinS ( int n, long long a[], long long b[], int mapno)

 {
      struct {
            long long al[IBANK];
      } banka;
      struct {
            long long bl[IBANK];
        } bankb;

      long long *al     = banka.al;
      long long *bl     = bankb.bl;
      int i, nbytes;
      /* nbytes = n*8;*/
      nbytes = (((n+3)/4)*4)*8;

      cm2obm_a(al, a, nbytes);
      wait_server_a();

      for (i = 0; i < n; i++) {
            Four_Bin(al[i], &bl[i]);
        }

      obm2cm_b (b, bl, nbytes);
      wait_server_b();

    }
```

D. 4 BIN .C FILE

```
/* main.c */
#include <stdio.h>
#include <sys/types.h>
#include <libmap.h>

#define SAMPLE_MAX 500000     /* Maximum number of phase samples. */
#define PADDING 17 /* number of padding sets before and after the sam-
ples */
void FourBinS();
void *Cache_Aligned_Allocate();
void Cache_Aligned_Free();

int main () {
      int i, nmap, mapnum, numofsamps, nbytes;
      short phzsampdat[SAMPLE_MAX], dummysample;
      FILE *fileptr;
      long I0, Q0, OtherBinDataSIN, OtherBinDataCOS;
      char phzincdat[16], ampscaldat[16], URB[16];
      char PRB, UNP, PSVin, ODVin, ODVout0, PSVout0, DRFM0, binnumber;
      long long temp, binprogram;
      long long* dataa;
      long long* datab;

/* Timing variables. */
      double tstart, tend, tcume, ttotal;
      extern double second();

/* initialization */
      tstart = second();
      mapnum = 0;
      nmap = 1;
      numofsamps=0;
      dummysample=0;

/* Read in phase increment values. */
      if ((fileptr = fopen("datafiles/phzinc.txt", "r")) == NULL)
```

```
            fprintf(stderr, "\n\nTERMINAL FAULT:  File phzinc.txt not
found.\n\n");
      binnumber = 0;
      while (fscanf(fileptr, "%x", &phzincdat[binnumber]) != EOF)
      {
            binnumber++;
      }
      fclose(fileptr);

/* Read in amplitude scaling values */
      if ((fileptr = fopen("datafiles/ampscal.txt", "r")) == NULL)
            fprintf(stderr, "\n\nTERMINAL FAULT:  File ampscal.txt not
found.\n\n");
      binnumber = 0;
      while (fscanf(fileptr, "%x", &ampscaldat[binnumber]) != EOF)
      {
            binnumber++;
      }
      fclose(fileptr);

/* Read in pulse phase samples */
      if ((fileptr = fopen("datafiles/phzsamp.txt", "r")) == NULL)
            fprintf(stderr, "\n\nTERMINAL FAULT:  File phzsamp.txt not
found.\n\n");
      numofsamps = 0;
      while (fscanf(fileptr, "%x", &phzsampdat[numofsamps]) != EOF)
      {
            numofsamps++;
      }
      fclose(fileptr);

    tend = second();
    tcume = tend - tstart;
    ttotal = tcume;
    printf ("\n Number of input samples: %d", numofsamps);
    printf ("\n Time for disk access of input data:  %19.10f", tcume);

      tstart = second();
      nbytes = (((numofsamps+PADDING*2+4)/4)*4)*8;
      dataa=Cache_Aligned_Allocate(nbytes);
      datab=Cache_Aligned_Allocate(nbytes);

    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time to allocate the data caches for the MAP:
%19.10f", tcume);

    tstart = second();
/* pack the data as follows:
Input: formats, cycle begins when PRB=1, 1 format per clock
 ten bits of each bin: 10 |  9-5   |4-1 |
                       URB|PhaseInc|Gain|
 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
unused | bin3 | bin2 | bin1 | bin0 |sample |PSVin |UNP |PRB
 63-8                              | 7-3   | 2    | 1  | 0
unused                             |sample |PSVin |UNP |PRB
```

```
-- The first sample can be entered on clock 2 if PSVin and UNP are
taken high
*/

      for (i=0; i<16; i++){
            URB[i]=1;}/*set all bins to be used*/
      for (i=0; i<PADDING; i++){/*pad the control signals before and
after*/
            dataa[i]=0; /*set PSVin, UNP, and PRB = 000 */
            dataa[i+numofsamps+PADDING+1]=0; /*set PSVin, UNP, and PRB
= 000 */
            }

            PSVin=0;
            UNP=0;
            PRB=1; /* start the program sequence*/
            temp=((long long) URB[3] & 0x1LL);
            temp=temp<<5 | ((long long) phzincdat[3] & 0x1FLL);
            temp=temp<<4 | ((long long) ampscaldat[3] & 0xFLL);
            temp=temp<<1 | ((long long) URB[2] & 0x1LL);
            temp=temp<<5 | ((long long) phzincdat[2] & 0x1FLL);
            temp=temp<<4 | ((long long) ampscaldat[2] & 0xFLL);
            temp=temp<<1 | ((long long) URB[1] & 0x1LL);
            temp=temp<<5 | ((long long) phzincdat[1] & 0x1FLL);
            temp=temp<<4 | ((long long) ampscaldat[1] & 0xFLL);
            temp=temp<<1 | ((long long) URB[0] & 0x1LL);
            temp=temp<<5 | ((long long) phzincdat[0] & 0x1FLL);
            temp=temp<<4 | ((long long) ampscaldat[0] & 0xFLL);
            temp=temp<<5 | ((long long) dummysample & 0x1FLL);
            temp=temp<<1 | ((long long) PSVin & 0x1LL);
            temp=temp<<1 | ((long long) UNP & 0x1LL);
            temp=temp<<1 | ((long long) PRB & 0x1LL);
            dataa[PADDING]=temp;

            PRB=0;
            PSVin=1; /* start sample input */
            UNP=1; /* use new programming */
            temp=((long long) phzsampdat[0] & 0x1FLL);
            temp=temp<<1 | ((long long) PSVin & 0x1LL);
            temp=temp<<1 | ((long long) UNP & 0x1LL);
            temp=temp<<1 | ((long long) PRB & 0x1LL);
            dataa[1+PADDING]=temp;

            UNP=0;

      for (i = 1; i < numofsamps; i++) {
            temp=((long long) phzsampdat[i] & 0x1FLL);
            temp=temp<<1 | ((long long) PSVin & 0x1LL);
            temp=temp<<1 | ((long long) UNP & 0x1LL);
            temp=temp<<1 | ((long long) PRB & 0x1LL);
            dataa[i+PADDING+1]=temp;}

    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time to pack the data for transfer to MAP:  %19.10f",
tcume);
```

```
    tstart = second();
/* allocate map to this problem */
      if (map_allocate (nmap)) {
            fprintf (stdout, "Map allocation failed.\n");
      exit (1);
      }
    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time for MAP allocation:  %19.10f", tcume);

    tstart = second();
/* call compute */
      FourBinS (numofsamps+PADDING*2+1, dataa, datab, mapnum);
    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time for MAP call:  %19.10f", tcume);

    tstart = second();
/* Open output file for writing. */
      if ((fileptr = fopen("datafiles/IandQout.txt", "w")) == NULL)
                  fprintf(stderr, "\n\nTERMINAL FAULT:  File
IandQout.txt cannot be written.\n\n");

/* put headers in output file */
      fprintf(fileptr, "Iout  Qout  ODVout PSVout DRFM\n");
      fprintf(fileptr, "----- ----- ------ ------ -----\n");

/* unpack the results and send to output*/
      for (i = 0; i < numofsamps+PADDING*2+1; i++) {
            DRFM0=datab[i] & 0x1FLL;
            I0=datab[i]>>5 & 0x1FFFFLL;
            Q0=datab[i]>>22 & 0x1FFFFLL;
            ODVout0=datab[i]>>39 & 0x1LL;
            PSVout0=datab[i]>>40 & 0x1LL;
            fprintf(fileptr, "%05X %05X   %01X      %01X    %02X\n",
I0, Q0, ODVout0, PSVout0, DRFM0);
      }
   fclose (fileptr);

   tend = second();
   tcume = tend - tstart;
   ttotal = ttotal + tcume;
   printf ("\n Time to unpack results and send to output file:
%19.10f", tcume);

   tstart = second();
/* free the map */
      if (map_free (nmap)) {
             printf ("Map deallocation failed. \n");
        exit (1);
        }

    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time to free the MAP:  %19.10f", tcume);

      tstart = second();

      Cache_Aligned_Free((char *)dataa);
      Cache_Aligned_Free((char *)datab);

    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time to free the data arrays:  %19.10f", tcume);
    printf ("\n Total Time: %19.10f\n\n", ttotal);
}
```

E. 8 BIN .BOX FILE

```
module EightBin (Input, Output, CLK) /* synthesis syn_black_box */ ;
      input [63:0] Input;
      output [63:0] Output;
      input CLK;

endmodule
```

F. 8 BIN .INFO FILE

```
BEGIN_DEF "Eight_Bin"
      MACRO = "EightBin";
      STATEFUL = NO;
      EXTERNAL = NO;
      PIPELINED = YES;
      LATENCY = 21;

      INPUTS = 1:
      I0 = INT 64 BITS (Input[63:0])
      ;

      OUTPUTS = 1:
      O0 = INT 64 BITS (Output[63:0])
      ;

      IN_SIGNAL : 1 BITS "CLK"="CLOCK";

END_DEF
```

G. 8 BIN .MC FILE

```
/* EightBinS.mc */
#include <libmap.h>

#define IBANK MAX_OBM_SIZE

void EightBinS ( int n, long long a[], long long b[], int mapno)

 {
      struct {
            long long al[IBANK];
      } banka;
      struct {
            long long bl[IBANK];
        } bankb;

      long long *al     = banka.al;
      long long *bl     = bankb.bl;


      int i, nbytes;
      /* nbytes = n*8;*/
      nbytes = (((n+3)/4)*4)*8;

      cm2obm_a(al, a, nbytes);
      wait_server_a();

      for (i = 0; i < n; i++) {
            Eight_Bin(al[i], &bl[i]);
        }

      obm2cm_b (b, bl, nbytes);
      wait_server_b();

    }
```

H. 8 BIN .C FILE

```
/* main.c */

#include <stdio.h>
#include <sys/types.h>
#include <libmap.h>

#define SAMPLE_MAX 500000     /* Maximum number of phase samples. */
#define PADDING 17 /* number of padding sets before and after the sam-
ples */
void EightBinS();
void *Cache_Aligned_Allocate();
void Cache_Aligned_Free();

int main () {

      int i, nmap, mapnum, numofsamps, nbytes;
      short phzsampdat[SAMPLE_MAX],dummysample;
      FILE *fileptr;
      long I0, Q0, OtherBinDataSIN, OtherBinDataCOS;
      char phzincdat[16], ampscaldat[16], URB[16];
      char PRB, UNP, PSVin, ODVin, ODVout0, PSVout0, DRFM0, binnumber;
      long long temp, binprogram;
      long long* dataa;
      long long* datab;

/* Timing variables. */
      double tstart, tend, tcume, ttotal;
      extern double second();

/* initialization */
```

```
      tstart = second();
      mapnum = 0;
      nmap = 1;
      numofsamps=0;
      dummysample=0;

/* Read in phase increment values. */
      if ((fileptr = fopen("datafiles/phzinc.txt", "r")) == NULL)
            fprintf(stderr, "\n\nTERMINAL FAULT:  File phzinc.txt not
found.\n\n");
      binnumber = 0;
      while (fscanf(fileptr, "%x", &phzincdat[binnumber]) != EOF)
      {
            binnumber++;
      }
      fclose(fileptr);

/* Read in amplitude scaling values */
      if ((fileptr = fopen("datafiles/ampscal.txt", "r")) == NULL)
            fprintf(stderr, "\n\nTERMINAL FAULT:  File ampscal.txt not
found.\n\n");
      binnumber = 0;
      while (fscanf(fileptr, "%x", &ampscaldat[binnumber]) != EOF)
      {
            binnumber++;
      }
      fclose(fileptr);

/* Read in pulse phase samples */
      if ((fileptr = fopen("datafiles/phzsamp.txt", "r")) == NULL)
            fprintf(stderr, "\n\nTERMINAL FAULT:  File phzsamp.txt not
found.\n\n");
      numofsamps = 0;
      while (fscanf(fileptr, "%x", &phzsampdat[numofsamps]) != EOF)
      {
            numofsamps++;
      }
      fclose(fileptr);

    tend = second();
    tcume = tend - tstart;
    ttotal = tcume;
    printf ("\n Number of input samples: %d", numofsamps);
    printf ("\n Time for disk access of input data:  %19.10f", tcume);

      tstart = second();
      nbytes = (((numofsamps+PADDING*2+2)/4)*4)*8;
      dataa=Cache_Aligned_Allocate(nbytes);
      datab=Cache_Aligned_Allocate(nbytes);

    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time to allocate the data caches for the MAP:
%19.10f", tcume);

    tstart = second();
```

```
/* pack the data as follows:
Input: formats, cycle begins when PRB=1, 1 format per clock
 ten bits of each bin: 10 |  9-5   |4-1 |
                      URB|PhaseInc|Gain|
 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
unused | bin3 | bin2 | bin1 | bin0 |sample |PSVin |UNP |PRB
 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
unused | bin7 | bin6 | bin5 | bin4 |sample |PSVin |UNP |PRB
 63-8                              | 7-3   | 2    | 1  | 0
unused                             |sample |PSVin |UNP |PRB
-- The first sample can be entered on clock 3 if PSVin and UNP are
taken high
*/

      for (i=0; i<16; i++){
            URB[i]=1;}/*set all bins to be used*/
      for (i=0; i<PADDING; i++){/*pad the control signals before and
after*/
            dataa[i]=0; /*set PSVin, UNP, and PRB = 000 */
            dataa[i+numofsamps+PADDING+2]=0; /*set PSVin, UNP, and PRB
= 000 */
            }

            PSVin=0;
            UNP=0;
            PRB=1; /* start the program sequence*/
            temp=((long long) URB[3] & 0x1LL);
            temp=temp<<5 | ((long long) phzincdat[3] & 0x1FLL);
            temp=temp<<4 | ((long long) ampscaldat[3] & 0xFLL);
            temp=temp<<1 | ((long long) URB[2] & 0x1LL);
            temp=temp<<5 | ((long long) phzincdat[2] & 0x1FLL);
            temp=temp<<4 | ((long long) ampscaldat[2] & 0xFLL);
            temp=temp<<1 | ((long long) URB[1] & 0x1LL);
            temp=temp<<5 | ((long long) phzincdat[1] & 0x1FLL);
            temp=temp<<4 | ((long long) ampscaldat[1] & 0xFLL);
            temp=temp<<1 | ((long long) URB[0] & 0x1LL);
            temp=temp<<5 | ((long long) phzincdat[0] & 0x1FLL);
            temp=temp<<4 | ((long long) ampscaldat[0] & 0xFLL);
            temp=temp<<5 | ((long long) dummysample & 0x1FLL);
            temp=temp<<1 | ((long long) PSVin & 0x1LL);
            temp=temp<<1 | ((long long) UNP & 0x1LL);
            temp=temp<<1 | ((long long) PRB & 0x1LL);
            dataa[PADDING]=temp;

            PRB=0;
            temp=((long long) URB[7] & 0x1LL);
            temp=temp<<5 | ((long long) phzincdat[7] & 0x1FLL);
            temp=temp<<4 | ((long long) ampscaldat[7] & 0xFLL);
            temp=temp<<1 | ((long long) URB[6] & 0x1LL);
            temp=temp<<5 | ((long long) phzincdat[6] & 0x1FLL);
            temp=temp<<4 | ((long long) ampscaldat[6] & 0xFLL);
            temp=temp<<1 | ((long long) URB[5] & 0x1LL);
            temp=temp<<5 | ((long long) phzincdat[5] & 0x1FLL);
            temp=temp<<4 | ((long long) ampscaldat[5] & 0xFLL);
            temp=temp<<1 | ((long long) URB[4] & 0x1LL);
            temp=temp<<5 | ((long long) phzincdat[4] & 0x1FLL);
            temp=temp<<4 | ((long long) ampscaldat[4] & 0xFLL);
            temp=temp<<5 | ((long long) dummysample & 0x1FLL);
            temp=temp<<1 | ((long long) PSVin & 0x1LL);
            temp=temp<<1 | ((long long) UNP & 0x1LL);
            temp=temp<<1 | ((long long) PRB & 0x1LL);
            dataa[1+PADDING]=temp;

            PSVin=1; /* start sample input */
            UNP=1; /* use new programming */
            temp=((long long) phzsampdat[0] & 0x1FLL);
            temp=temp<<1 | ((long long) PSVin & 0x1LL);
            temp=temp<<1 | ((long long) UNP & 0x1LL);
            temp=temp<<1 | ((long long) PRB & 0x1LL);
            dataa[2+PADDING]=temp;

            UNP=0;

      for (i = 1; i < numofsamps; i++) {
            temp=((long long) phzsampdat[i] & 0x1FLL);
            temp=temp<<1 | ((long long) PSVin & 0x1LL);
            temp=temp<<1 | ((long long) UNP & 0x1LL);
            temp=temp<<1 | ((long long) PRB & 0x1LL);
            dataa[i+PADDING+2]=temp;}

    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time to pack the data for transfer to MAP:  %19.10f",
tcume);

    tstart = second();
/* allocate map to this problem */
      if (map_allocate (nmap)) {
            fprintf (stdout, "Map allocation failed.\n");
      exit (1);
      }
    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time for MAP allocation:  %19.10f", tcume);

    tstart = second();
/* call compute */
      EightBinS (numofsamps+PADDING*2+2, dataa, datab, mapnum);
    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time for MAP call:  %19.10f", tcume);

    tstart = second();
/* Open output file for writing. */
      if ((fileptr = fopen("datafiles/IandQout.txt", "w")) == NULL)
                  fprintf(stderr, "\n\nTERMINAL FAULT:  File
IandQout.txt cannot be written.\n\n");

/* put headers in output file */
      fprintf(fileptr, "Iout  Qout  ODVout PSVout DRFM\n");
      fprintf(fileptr, "----- ----- ------ ------ -----\n");
```

```
/* unpack the results and send to output*/
      for (i = 0; i < numofsamps+PADDING*2+2; i++) {
            DRFM0=datab[i] & 0x1FLL;
            I0=datab[i]>>5 & 0x1FFFFLL;
            Q0=datab[i]>>22 & 0x1FFFFLL;
            ODVout0=datab[i]>>39 & 0x1LL;
            PSVout0=datab[i]>>40 & 0x1LL;
            fprintf(fileptr, "%05X %05X   %01X      %01X    %02X\n",
I0, Q0, ODVout0, PSVout0, DRFM0);
      }
   fclose (fileptr);

   tend = second();
   tcume = tend - tstart;
   ttotal = ttotal + tcume;
   printf ("\n Time to unpack results and send to output file:
%19.10f", tcume);

   tstart = second();
/* free the map */
      if (map_free (nmap)) {
             printf ("Map deallocation failed. \n");
        exit (1);
        }

    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time to free the MAP:  %19.10f", tcume);

      tstart = second();

      Cache_Aligned_Free((char *)dataa);
      Cache_Aligned_Free((char *)datab);

    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time to free the data arrays:  %19.10f", tcume);
    printf ("\n Total Time: %19.10f\n\n", ttotal);

}
```

I. 16 BIN .BOX FILE

```
module SixteenBin (Input, Output, CLK) /* synthesis syn_black_box */ ;
      input [63:0] Input;
      output [63:0] Output;
      input CLK;

endmodule
```

J. 16 BIN .INFO FILE

```
BEGIN_DEF "Sixteen_Bin"
      MACRO = "SixteenBin";
      STATEFUL = NO;
      EXTERNAL = NO;
      PIPELINED = YES;
      LATENCY = 21;

      INPUTS = 1:
      I0 = INT 64 BITS (Input[63:0])
      ;

      OUTPUTS = 1:
      O0 = INT 64 BITS (Output[63:0])
      ;

      IN_SIGNAL : 1 BITS "CLK"="CLOCK";
END_DEF
```

K. 16 BIN .MC FILE

```
/* SixteenBinS.mc */
#include <libmap.h>

#define IBANK MAX_OBM_SIZE

void SixteenBinS ( int n, long long a[], long long b[], int mapno)

 {
      struct {
            long long al[IBANK];
      } banka;
      struct {
            long long bl[IBANK];
        } bankb;

      long long *al     = banka.al;
      long long *bl     = bankb.bl;


      int i, nbytes;
      /* nbytes = n*8;*/
      nbytes = (((n+3)/4)*4)*8;

      cm2obm_a(al, a, nbytes);
      wait_server_a();

      for (i = 0; i < n; i++) {
            Sixteen_Bin(al[i], &bl[i]);
        }

      obm2cm_b (b, bl, nbytes);
      wait_server_b();
    }
```

L. 16 BIN .C FILE

```
/* main.c */
```

```
#include <stdio.h>
#include <sys/types.h>
#include <libmap.h>

#define SAMPLE_MAX 500000     /* Maximum number of phase samples. */
#define PADDING 17 /* number of padding sets before and after the sam-
ples */
void SixteenBinS();
void *Cache_Aligned_Allocate();
void Cache_Aligned_Free();

int main () {

      int i, nmap, mapnum, numofsamps, nbytes;
      short phzsampdat[SAMPLE_MAX], dummysample;
      FILE *fileptr;
      long I0, Q0, OtherBinDataSIN, OtherBinDataCOS;
      char phzincdat[16], ampscaldat[16], URB[16];
      char UNP, PRB, PSVin, ODVin, ODVout0, PSVout0, DRFM0, binnumber;
      long long temp, binprogram;
      long long* dataa;
      long long* datab;

/* Timing variables. */
      double tstart, tend, tcume, ttotal;
      extern double second();

/* initialization */
      tstart = second();
      mapnum = 0;
      nmap = 1;
      numofsamps=0;
      dummysample=0;

/* Read in phase increment values. */
      if ((fileptr = fopen("datafiles/phzinc.txt", "r")) == NULL)
            fprintf(stderr, "\n\nTERMINAL FAULT:  File phzinc.txt not
found.\n\n");
      binnumber = 0;
      while (fscanf(fileptr, "%x", &phzincdat[binnumber]) != EOF)
      {
            binnumber++;
      }
      fclose(fileptr);

/* Read in amplitude scaling values */
      if ((fileptr = fopen("datafiles/ampscal.txt", "r")) == NULL)
            fprintf(stderr, "\n\nTERMINAL FAULT:  File ampscal.txt not
found.\n\n");
      binnumber = 0;
      while (fscanf(fileptr, "%x", &ampscaldat[binnumber]) != EOF)
      {
            binnumber++;
      }
      fclose(fileptr);
```

```
/* Read in pulse phase samples */
      if ((fileptr = fopen("datafiles/phzsamp.txt", "r")) == NULL)
            fprintf(stderr, "\n\nTERMINAL FAULT:  File phzsamp.txt not
found.\n\n");
      numofsamps = 0;
      while (fscanf(fileptr, "%x", &phzsampdat[numofsamps]) != EOF)
      {
            numofsamps++;
      }
      fclose(fileptr);

    tend = second();
    tcume = tend - tstart;
    ttotal = tcume;
    printf ("\n Number of input samples: %d", numofsamps);
    printf ("\n Time for disk access of input data:  %19.10f", tcume);

      tstart = second();
      nbytes = (((numofsamps+PADDING*2+4)/4)*4)*8;
      dataa=Cache_Aligned_Allocate(nbytes);
      datab=Cache_Aligned_Allocate(nbytes);

    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time to allocate the data caches for the MAP:
%19.10f", tcume);

    tstart = second();
/* pack the data as follows:
Input: formats, cycle begins when PRB=1, 1 format per clock
 ten bits of each bin: 10 |  9-5    |4-1 |
                       URB|PhaseInc|Gain|
 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
unused | bin3 | bin2 | bin1 | bin0 |sample |PSVin |UNP |PRB
 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
unused | bin7 | bin6 | bin5 | bin4 |sample |PSVin |UNP |PRB
 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
unused | bin11| bin10| bin9 | bin8 |sample |PSVin |UNP |PRB
 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
unused | bin15| bin14| bin13| bin12|sample |PSVin |UNP |PRB
 63-8                             | 7-3   | 2    | 1  | 0
unused                            |sample |PSVin |UNP |PRB
-- The first sample can be entered on clock 5 if PSVin and UNP are
taken high
*/

      for (i=0; i<16; i++){
            URB[i]=1;}/*set all bins to be used*/
      for (i=0; i<PADDING; i++){/*pad the control signals before and
after*/
            dataa[i]=0; /*set PSVin, UNP, and PRB = 000 */
            dataa[i+numofsamps+PADDING+4]=0; /*set PSVin, UNP, and PRB
= 000 */
            }

            PSVin=0;
```

```
UNP=0;
PRB=1; /* start the program sequence*/
temp=((long long) URB[3] & 0x1LL);
temp=temp<<5 | ((long long) phzincdat[3] & 0x1FLL);
temp=temp<<4 | ((long long) ampscaldat[3] & 0xFLL);
temp=temp<<1 | ((long long) URB[2] & 0x1LL);
temp=temp<<5 | ((long long) phzincdat[2] & 0x1FLL);
temp=temp<<4 | ((long long) ampscaldat[2] & 0xFLL);
temp=temp<<1 | ((long long) URB[1] & 0x1LL);
temp=temp<<5 | ((long long) phzincdat[1] & 0x1FLL);
temp=temp<<4 | ((long long) ampscaldat[1] & 0xFLL);
temp=temp<<1 | ((long long) URB[0] & 0x1LL);
temp=temp<<5 | ((long long) phzincdat[0] & 0x1FLL);
temp=temp<<4 | ((long long) ampscaldat[0] & 0xFLL);
temp=temp<<5 | ((long long) dummysample & 0x1FLL);
temp=temp<<1 | ((long long) PSVin & 0x1LL);
temp=temp<<1 | ((long long) UNP & 0x1LL);
temp=temp<<1 | ((long long) PRB & 0x1LL);
dataa[PADDING]=temp;

PRB=0;
temp=((long long) URB[7] & 0x1LL);
temp=temp<<5 | ((long long) phzincdat[7] & 0x1FLL);
temp=temp<<4 | ((long long) ampscaldat[7] & 0xFLL);
temp=temp<<1 | ((long long) URB[6] & 0x1LL);
temp=temp<<5 | ((long long) phzincdat[6] & 0x1FLL);
temp=temp<<4 | ((long long) ampscaldat[6] & 0xFLL);
temp=temp<<1 | ((long long) URB[5] & 0x1LL);
temp=temp<<5 | ((long long) phzincdat[5] & 0x1FLL);
temp=temp<<4 | ((long long) ampscaldat[5] & 0xFLL);
temp=temp<<1 | ((long long) URB[4] & 0x1LL);
temp=temp<<5 | ((long long) phzincdat[4] & 0x1FLL);
temp=temp<<4 | ((long long) ampscaldat[4] & 0xFLL);
temp=temp<<5 | ((long long) dummysample & 0x1FLL);
temp=temp<<1 | ((long long) PSVin & 0x1LL);
temp=temp<<1 | ((long long) UNP & 0x1LL);
temp=temp<<1 | ((long long) PRB & 0x1LL);
dataa[1+PADDING]=temp;

temp=((long long) URB[11] & 0x1LL);
temp=temp<<5 | ((long long) phzincdat[11] & 0x1FLL);
temp=temp<<4 | ((long long) ampscaldat[11] & 0xFLL);
temp=temp<<1 | ((long long) URB[10] & 0x1LL);
temp=temp<<5 | ((long long) phzincdat[10] & 0x1FLL);
temp=temp<<4 | ((long long) ampscaldat[10] & 0xFLL);
temp=temp<<1 | ((long long) URB[9] & 0x1LL);
temp=temp<<5 | ((long long) phzincdat[9] & 0x1FLL);
temp=temp<<4 | ((long long) ampscaldat[9] & 0xFLL);
temp=temp<<1 | ((long long) URB[8] & 0x1LL);
temp=temp<<5 | ((long long) phzincdat[8] & 0x1FLL);
temp=temp<<4 | ((long long) ampscaldat[8] & 0xFLL);
temp=temp<<5 | ((long long) dummysample & 0x1FLL);
temp=temp<<1 | ((long long) PSVin & 0x1LL);
temp=temp<<1 | ((long long) UNP & 0x1LL);
temp=temp<<1 | ((long long) PRB & 0x1LL);
dataa[2+PADDING]=temp;
```

```
            temp=((long long) URB[15] & 0x1LL);
            temp=temp<<5 | ((long long) phzincdat[15] & 0x1FLL);
            temp=temp<<4 | ((long long) ampscaldat[15] & 0xFLL);
            temp=temp<<1 | ((long long) URB[14] & 0x1LL);
            temp=temp<<5 | ((long long) phzincdat[14] & 0x1FLL);
            temp=temp<<4 | ((long long) ampscaldat[14] & 0xFLL);
            temp=temp<<1 | ((long long) URB[13] & 0x1LL);
            temp=temp<<5 | ((long long) phzincdat[13] & 0x1FLL);
            temp=temp<<4 | ((long long) ampscaldat[13] & 0xFLL);
            temp=temp<<1 | ((long long) URB[12] & 0x1LL);
            temp=temp<<5 | ((long long) phzincdat[12] & 0x1FLL);
            temp=temp<<4 | ((long long) ampscaldat[12] & 0xFLL);
            temp=temp<<5 | ((long long) dummysample & 0x1FLL);
            temp=temp<<1 | ((long long) PSVin & 0x1LL);
            temp=temp<<1 | ((long long) UNP & 0x1LL);
            temp=temp<<1 | ((long long) PRB & 0x1LL);
            dataa[3+PADDING]=temp;

            PSVin=1; /* start sample input */
            UNP=1; /* use new programming */
            temp=((long long) phzsampdat[0] & 0x1FLL);
            temp=temp<<1 | ((long long) PSVin & 0x1LL);
            temp=temp<<1 | ((long long) UNP & 0x1LL);
            temp=temp<<1 | ((long long) PRB & 0x1LL);
            dataa[4+PADDING]=temp;

            UNP=0;
      for (i = 1; i < numofsamps; i++) {
            temp=((long long) phzsampdat[i] & 0x1FLL);
            temp=temp<<1 | ((long long) PSVin & 0x1LL);
            temp=temp<<1 | ((long long) UNP & 0x1LL);
            temp=temp<<1 | ((long long) PRB & 0x1LL);
            dataa[i+PADDING+4]=temp;}

    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time to pack the data for transfer to MAP:  %19.10f",
tcume);

    tstart = second();
/* allocate map to this problem */
      if (map_allocate (nmap)) {
            fprintf (stdout, "Map allocation failed.\n");
      exit (1);
      }
    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time for MAP allocation:  %19.10f", tcume);

    tstart = second();
/* call compute */
      SixteenBinS (numofsamps+PADDING*2+4, dataa, datab, mapnum);
    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time for MAP call:  %19.10f", tcume);

    tstart = second();
/* Open output file for writing. */
      if ((fileptr = fopen("datafiles/IandQout.txt", "w")) == NULL)
                  fprintf(stderr, "\n\nTERMINAL FAULT:  File
IandQout.txt cannot be written.\n\n");

/* put headers in output file */
      fprintf(fileptr, "Iout  Qout  ODVout PSVout DRFM\n");
      fprintf(fileptr, "----- ----- ------ ------ -----\n");

/* unpack the results and send to output*/
      for (i = 0; i < numofsamps+PADDING*2+4; i++) {
            DRFM0=datab[i] & 0x1FLL;
            I0=datab[i]>>5 & 0x1FFFFLL;
            Q0=datab[i]>>22 & 0x1FFFFLL;
            ODVout0=datab[i]>>39 & 0x1LL;
            PSVout0=datab[i]>>40 & 0x1LL;
            fprintf(fileptr, "%05X %05X   %01X      %01X    %02X\n",
I0, Q0, ODVout0, PSVout0, DRFM0);
      }
   fclose (fileptr);

   tend = second();
   tcume = tend - tstart;
   ttotal = ttotal + tcume;
   printf ("\n Time to unpack results and send to output file:
%19.10f", tcume);

   tstart = second();
/* free the map */
      if (map_free (nmap)) {
             printf ("Map deallocation failed. \n");
        exit (1);
        }

    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time to free the MAP:  %19.10f", tcume);

      tstart = second();

      Cache_Aligned_Free((char *)dataa);
      Cache_Aligned_Free((char *)datab);

    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time to free the data arrays:  %19.10f", tcume);
    printf ("\n Total Time: %19.10f\n\n", ttotal);

}
```

M.   64 BIN .BOX FILE

```
module SixtyFourBin (Input, Output, CLK) /* synthesis syn_black_box */
;
      input [63:0] Input;
      output [63:0] Output;
      input CLK;
endmodule
```

N.   64 BIN .INFO FILE

```
BEGIN_DEF "SixtyFour_Bin"
      MACRO = "SixtyFourBin";
      STATEFUL = NO;
      EXTERNAL = NO;
      PIPELINED = YES;
      LATENCY = 133;

      INPUTS = 1:
      I0 = INT 64 BITS (Input[63:0])
      ;

      OUTPUTS = 1:
      O0 = INT 64 BITS (Output[63:0])
      ;

      IN_SIGNAL : 1 BITS "CLK"="CLOCK";


END_DEF
```

O.   64 BIN .MC FILE

```
/* SixtyFourBinS.mc */
#include <libmap.h>

#define IBANK MAX_OBM_SIZE

void SixtyFourBinS ( int n, long long a[], long long b[], int mapno)

 {
      struct {
            long long al[IBANK];
      } banka;
      struct {
            long long bl[IBANK];
        } bankb;

      long long *al     = banka.al;
      long long *bl     = bankb.bl;


      int i, nbytes;
      /* nbytes = n*8;*/
      nbytes = (((n+3)/4)*4)*8;

      cm2obm_a(al, a, nbytes);
      wait_server_a();
      for (i = 0; i < n; i++) {
            SixtyFour_Bin(al[i], &bl[i]);
        }

      obm2cm_b (b, bl, nbytes);
      wait_server_b();

    }
```

P. 64 BIN .C FILE

```
/* main.c */

#include <stdio.h>
#include <sys/types.h>
#include <libmap.h>

#define SAMPLE_MAX 500000     /* Maximum number of phase samples. */
#define PADDING 64 /* number of padding sets before and after the sam-
ples */
void SixtyFourBinS();
void *Cache_Aligned_Allocate();
void Cache_Aligned_Free();

int main () {

      int i, nmap, mapnum, numofsamps, nbytes;
      short phzsampdat[SAMPLE_MAX], dummysample;
      FILE *fileptr;
      long I0, Q0, OtherBinDataSIN, OtherBinDataCOS;
      char phzincdat[64], ampscaldat[64], URB[64];
      char UNP, PRB, PSVin, ODVin, ODVout0, PSVout0, DRFM0, binnumber;
      long long temp, binprogram;
      long long* dataa;
      long long* datab;

/* Timing variables. */
      double tstart, tend, tcume, ttotal;
      extern double second();

/* initialization */
      tstart = second();
      mapnum = 0;
      nmap = 1;
      numofsamps=0;
      dummysample=0;

/* Read in phase increment values. */
      if ((fileptr = fopen("datafiles/phzinc.txt", "r")) == NULL)
            fprintf(stderr, "\n\nTERMINAL FAULT:  File phzinc.txt not
found.\n\n");
      binnumber = 0;
      while (fscanf(fileptr, "%x", &phzincdat[binnumber]) != EOF)
      {
            binnumber++;
      }
      fclose(fileptr);

/* Read in amplitude scaling values */
      if ((fileptr = fopen("datafiles/ampscal.txt", "r")) == NULL)
            fprintf(stderr, "\n\nTERMINAL FAULT:  File ampscal.txt not
found.\n\n");
      binnumber = 0;
      while (fscanf(fileptr, "%x", &ampscaldat[binnumber]) != EOF)
      {
            binnumber++;
      }
      fclose(fileptr);

/* Read in pulse phase samples */
      if ((fileptr = fopen("datafiles/phzsamp.txt", "r")) == NULL)
            fprintf(stderr, "\n\nTERMINAL FAULT:  File phzsamp.txt not
found.\n\n");
      numofsamps = 0;
      while (fscanf(fileptr, "%x", &phzsampdat[numofsamps]) != EOF)
      {
            numofsamps++;
      }
      fclose(fileptr);

    tend = second();
    tcume = tend - tstart;
    ttotal = tcume;
    printf ("\n Number of input samples: %d", numofsamps);
    printf ("\n Time for disk access of input data:  %19.10f", tcume);

      tstart = second();
      nbytes = (((numofsamps+PADDING*2+16)/4)*4)*8;
      dataa=Cache_Aligned_Allocate(nbytes);
      datab=Cache_Aligned_Allocate(nbytes);

    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time to allocate the data caches for the MAP:
%19.10f", tcume);

    tstart = second();
/* pack the data as follows:
Input: formats, cycle begins when PRB=1, 1 format per clock
 ten bits of each bin: 10 |  9-5   |4-1 |
                       URB|PhaseInc|Gain|
 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
unused | bin3 | bin2 | bin1 | bin0 |sample |PSVin |UNP |PRB
 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
unused | bin7 | bin6 | bin5 | bin4 |sample |PSVin |UNP |PRB
 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
unused | bin11| bin10| bin9 | bin8 |sample |PSVin |UNP |PRB
 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
unused | bin15| bin14| bin13| bin12|sample |PSVin |UNP |PRB
more in same format up to bin63...
 63-8                            | 7-3   | 2    | 1  | 0
unused                           |sample |PSVin |UNP |PRB
```

```
-- The first sample can be entered on clock 65 if PSVin and UNP are
taken high
*/

      for (i=0; i<64; i++){
            URB[i]=1;}/*set all bins to be used*/
      for (i=0; i<PADDING; i++){/*pad the control signals before and
after*/
            dataa[i]=0; /*set PSVin, UNP, and PRB = 000 */
            dataa[i+numofsamps+PADDING+16]=0; /*set PSVin, UNP, and PRB
= 000 */
            }
            PSVin=0;
            UNP=0;
            PRB=1; /* start the program sequence*/

      for (i=0;i<64;i=i+4){
            if (i>0) PRB=0;

            temp=((long long) URB[i+3] & 0x1LL);
            temp=temp<<5 | ((long long) phzincdat[i+3] & 0x1FLL);
            temp=temp<<4 | ((long long) ampscaldat[i+3] & 0xFLL);
            temp=temp<<1 | ((long long) URB[i+2] & 0x1LL);
            temp=temp<<5 | ((long long) phzincdat[i+2] & 0x1FLL);
            temp=temp<<4 | ((long long) ampscaldat[i+2] & 0xFLL);
            temp=temp<<1 | ((long long) URB[i+1] & 0x1LL);
            temp=temp<<5 | ((long long) phzincdat[i+1] & 0x1FLL);
            temp=temp<<4 | ((long long) ampscaldat[i+1] & 0xFLL);
            temp=temp<<1 | ((long long) URB[i] & 0x1LL);
            temp=temp<<5 | ((long long) phzincdat[i] & 0x1FLL);
            temp=temp<<4 | ((long long) ampscaldat[i] & 0xFLL);

            temp=temp<<5 | ((long long) dummysample & 0x1FLL);
            temp=temp<<1 | ((long long) PSVin & 0x1LL);
            temp=temp<<1 | ((long long) UNP & 0x1LL);
            temp=temp<<1 | ((long long) PRB & 0x1LL);
            dataa[PADDING+i/4]=temp;}

      PSVin=1; /* start sample input */
      UNP=1; /* use new programming */
      temp=((long long) phzsampdat[0] & 0x1FLL);

      temp=temp<<1 | ((long long) PSVin & 0x1LL);
      temp=temp<<1 | ((long long) UNP & 0x1LL);
      temp=temp<<1 | ((long long) PRB & 0x1LL);
      dataa[16+PADDING]=temp;

      UNP=0;
      for (i = 1; i < numofsamps; i++) {
            temp=((long long) phzsampdat[i] & 0x1FLL);
            temp=temp<<1 | ((long long) PSVin & 0x1LL);
            temp=temp<<1 | ((long long) UNP & 0x1LL);
            temp=temp<<1 | ((long long) PRB & 0x1LL);
            dataa[i+PADDING+16]=temp;}

    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time to pack the data for transfer to MAP:  %19.10f",
tcume);

    tstart = second();

/* allocate map to this problem */
      if (map_allocate (nmap)) {
            fprintf (stdout, "Map allocation failed.\n");
      exit (1);
      }

    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time for MAP allocation:  %19.10f", tcume);

    tstart = second();
/* call compute */
      SixtyFourBinS (numofsamps+PADDING*2+16, dataa, datab, mapnum);
    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time for MAP call:  %19.10f", tcume);

    tstart = second();
/* Open output file for writing. */
      if ((fileptr = fopen("datafiles/IandQout.txt", "w")) == NULL)
                  fprintf(stderr, "\n\nTERMINAL FAULT:  File
IandQout.txt cannot be written.\n\n");

/* put headers in output file */
      fprintf(fileptr, "Iout  Qout  ODVout PSVout DRFM\n");
      fprintf(fileptr, "----- ----- ------ ------ -----\n");

/* unpack the results and send to output*/
      for (i = 0; i < numofsamps+PADDING*2+16; i++) {
            DRFM0=datab[i] & 0x1FLL;
            I0=datab[i]>>5 & 0x1FFFFLL;
            Q0=datab[i]>>22 & 0x1FFFFLL;
            ODVout0=datab[i]>>39 & 0x1LL;
            PSVout0=datab[i]>>40 & 0x1LL;
            fprintf(fileptr, "%05X %05X   %01X      %01X    %02X\n",
I0, Q0, ODVout0, PSVout0, DRFM0);
      }
   fclose (fileptr);

   tend = second();
   tcume = tend - tstart;
   ttotal = ttotal + tcume;
   printf ("\n Time to unpack results and send to output file:
%19.10f", tcume);

   tstart = second();
/* free the map */
      if (map_free (nmap)) {
            printf ("Map deallocation failed. \n");
        exit (1);
        }

    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time to free the MAP:  %19.10f", tcume);

      tstart = second();

      Cache_Aligned_Free((char *)dataa);
      Cache_Aligned_Free((char *)datab);

    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time to free the data arrays:  %19.10f", tcume);
    printf ("\n Total Time: %19.10f\n\n", ttotal);
}
```

Q. 128 BIN .BOX FILE

```
module One28Bin (Input, Output, CLK) /* synthesis syn_black_box */ ;
      input [63:0] Input;
      output [63:0] Output;
      input CLK;

endmodule
```

R. 128 BIN .INFO FILE

```
BEGIN_DEF "One28_Bin"
      MACRO = "One28Bin";
      STATEFUL = NO;
      EXTERNAL = NO;
      PIPELINED = YES;
      LATENCY = 261;

      INPUTS = 1:
      I0 = INT 64 BITS (Input[63:0])
      ;

      OUTPUTS = 1:
      O0 = INT 64 BITS (Output[63:0])
      ;

      IN_SIGNAL : 1 BITS "CLK"="CLOCK";

END_DEF
```

S. 128 BIN .MC FILE

```
/* One28BinS.mc */
#include <libmap.h>

#define IBANK MAX_OBM_SIZE
```

```
void One28BinS ( int n, long long a[], long long b[], int mapno)

 {
      struct {
            long long al[IBANK];
      } banka;
      struct {
            long long bl[IBANK];
        } bankb;

      long long *al     = banka.al;
      long long *bl     = bankb.bl;


      int i, nbytes;
      /* nbytes = n*8;*/
      nbytes = (((n+3)/4)*4)*8;

      cm2obm_a(al, a, nbytes);
      wait_server_a();

      for (i = 0; i < n; i++) {
            One28_Bin(al[i], &bl[i]);
        }

      obm2cm_b (b, bl, nbytes);
      wait_server_b();
    }
```

T. 128 BIN .C FILE

```
/* main.c */

#include <stdio.h>
#include <sys/types.h>
#include <libmap.h>

#define SAMPLE_MAX 500000     /* Maximum number of phase samples. */
#define PADDING 128 /* number of padding sets before and after the sam-
ples */
void One28BinS();
void *Cache_Aligned_Allocate();
void Cache_Aligned_Free();

int main () {

      int i, nmap, mapnum, numofsamps, nbytes;
      short phzsampdat[SAMPLE_MAX], dummysample;
      FILE *fileptr;
      long I0, Q0, OtherBinDataSIN, OtherBinDataCOS;
      char phzincdat[128], ampscaldat[128], URB[128];
      char UNP, PRB, PSVin, ODVin, ODVout0, PSVout0, DRFM0, binnumber;
      long long temp, binprogram;
      long long* dataa;
      long long* datab;
```

```
/* Timing variables. */
      double tstart, tend, tcume, ttotal;
      extern double second();

/* initialization */
      tstart = second();
      mapnum = 0;
      nmap = 1;
      numofsamps=0;
      dummysample=0;

/* Read in phase increment values. */
      if ((fileptr = fopen("datafiles/phzinc.txt", "r")) == NULL)
            fprintf(stderr, "\n\nTERMINAL FAULT:  File phzinc.txt not
found.\n\n");
      binnumber = 0;
      while (fscanf(fileptr, "%x", &phzincdat[binnumber]) != EOF)
      {
            binnumber++;
      }
      fclose(fileptr);

/* Read in amplitude scaling values */
      if ((fileptr = fopen("datafiles/ampscal.txt", "r")) == NULL)
            fprintf(stderr, "\n\nTERMINAL FAULT:  File ampscal.txt not
found.\n\n");
      binnumber = 0;
      while (fscanf(fileptr, "%x", &ampscaldat[binnumber]) != EOF)
      {
            binnumber++;
      }
      fclose(fileptr);

/* Read in pulse phase samples */
      if ((fileptr = fopen("datafiles/phzsamp.txt", "r")) == NULL)
            fprintf(stderr, "\n\nTERMINAL FAULT:  File phzsamp.txt not
found.\n\n");
      numofsamps = 0;
      while (fscanf(fileptr, "%x", &phzsampdat[numofsamps]) != EOF)
      {
            numofsamps++;
      }
      fclose(fileptr);

    tend = second();
    tcume = tend - tstart;
    ttotal = tcume;
    printf ("\n Number of input samples: %d", numofsamps);
    printf ("\n Time for disk access of input data:  %19.10f", tcume);

      tstart = second();
      nbytes = (((numofsamps+PADDING*2+32)/4)*4)*8;
      dataa=Cache_Aligned_Allocate(nbytes);
      datab=Cache_Aligned_Allocate(nbytes);

    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time to allocate the data caches for the MAP:
%19.10f", tcume);

    tstart = second();
/* pack the data as follows:
Input: formats, cycle begins when PRB=1, 1 format per clock
 ten bits of each bin: 10 |  9-5   |4-1 |
                       URB|PhaseInc|Gain|
 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
unused | bin3 | bin2 | bin1 | bin0 |sample |PSVin |UNP |PRB
 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
unused | bin7 | bin6 | bin5 | bin4 |sample |PSVin |UNP |PRB
 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
unused | bin11| bin10| bin9 | bin8 |sample |PSVin |UNP |PRB
 63-48 |47-38 |37-28 |27-18 | 17-8 | 7-3   | 2    | 1  | 0
unused | bin15| bin14| bin13| bin12|sample |PSVin |UNP |PRB
more in same format up to bin127...
 63-8                              | 7-3   | 2    | 1  | 0
unused                             |sample |PSVin |UNP |PRB
-- The first sample can be entered on clock 129 if PSVin and UNP are
taken high
*/

      for (i=0; i<128; i++){
            URB[i]=1;}/*set all bins to be used*/
      for (i=0; i<PADDING; i++){/*pad the control signals before and
after*/
            dataa[i]=0; /*set PSVin, UNP, and PRB = 000 */
            dataa[i+numofsamps+PADDING+32]=0; /*set PSVin, UNP, and PRB
= 000 */
            }
            PSVin=0;
            UNP=0;
            PRB=1; /* start the program sequence*/

      for (i=0;i<128;i=i+4){
            if (i>0) PRB=0;

            temp=((long long) URB[i+3] & 0x1LL);
            temp=temp<<5 | ((long long) phzincdat[i+3] & 0x1FLL);
            temp=temp<<4 | ((long long) ampscaldat[i+3] & 0xFLL);
            temp=temp<<1 | ((long long) URB[i+2] & 0x1LL);
            temp=temp<<5 | ((long long) phzincdat[i+2] & 0x1FLL);
            temp=temp<<4 | ((long long) ampscaldat[i+2] & 0xFLL);
            temp=temp<<1 | ((long long) URB[i+1] & 0x1LL);
            temp=temp<<5 | ((long long) phzincdat[i+1] & 0x1FLL);
            temp=temp<<4 | ((long long) ampscaldat[i+1] & 0xFLL);
            temp=temp<<1 | ((long long) URB[i] & 0x1LL);
            temp=temp<<5 | ((long long) phzincdat[i] & 0x1FLL);
            temp=temp<<4 | ((long long) ampscaldat[i] & 0xFLL);

            temp=temp<<5 | ((long long) dummysample & 0x1FLL);
            temp=temp<<1 | ((long long) PSVin & 0x1LL);
            temp=temp<<1 | ((long long) UNP & 0x1LL);
            temp=temp<<1 | ((long long) PRB & 0x1LL);
            dataa[PADDING+i/4]=temp;}

      PSVin=1; /* start sample input */
      UNP=1; /* use new programming */
      temp=((long long) phzsampdat[0] & 0x1FLL);

      temp=temp<<1 | ((long long) PSVin & 0x1LL);
      temp=temp<<1 | ((long long) UNP & 0x1LL);
      temp=temp<<1 | ((long long) PRB & 0x1LL);
      dataa[32+PADDING]=temp;

      UNP=0;

      for (i = 1; i < numofsamps; i++) {
            temp=((long long) phzsampdat[i] & 0x1FLL);
            temp=temp<<1 | ((long long) PSVin & 0x1LL);
            temp=temp<<1 | ((long long) UNP & 0x1LL);
            temp=temp<<1 | ((long long) PRB & 0x1LL);
            dataa[i+PADDING+32]=temp;}

    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time to pack the data for transfer to MAP:  %19.10f",
tcume);

    tstart = second();

/* allocate map to this problem */
      if (map_allocate (nmap)) {
            fprintf (stdout, "Map allocation failed.\n");
      exit (1);
      }

    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time for MAP allocation:  %19.10f", tcume);

    tstart = second();
/* call compute */
      One28BinS (numofsamps+PADDING*2+32, dataa, datab, mapnum);
    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time for MAP call:  %19.10f", tcume);

    tstart = second();
/* Open output file for writing. */
      if ((fileptr = fopen("datafiles/IandQout.txt", "w")) == NULL)
                  fprintf(stderr, "\n\nTERMINAL FAULT:  File
IandQout.txt cannot be written.\n\n");

/* put headers in output file */
      fprintf(fileptr, "Iout  Qout  ODVout PSVout DRFM\n");
      fprintf(fileptr, "----- ----- ------ ------ -----\n");
```

```
/* unpack the results and send to output*/
      for (i = 0; i < numofsamps+PADDING*2+32; i++) {
            DRFM0=datab[i] & 0x1FLL;
            I0=datab[i]>>5 & 0x1FFFFLL;
            Q0=datab[i]>>22 & 0x1FFFFLL;
            ODVout0=datab[i]>>39 & 0x1LL;
            PSVout0=datab[i]>>40 & 0x1LL;
            fprintf(fileptr, "%05X %05X   %01X      %01X    %02X\n",
I0, Q0, ODVout0, PSVout0, DRFM0);
      }
   fclose (fileptr);

   tend = second();
   tcume = tend - tstart;
   ttotal = ttotal + tcume;
   printf ("\n Time to unpack results and send to output file:
%19.10f", tcume);

   tstart = second();
/* free the map */
      if (map_free (nmap)) {
             printf ("Map deallocation failed. \n");
        exit (1);
        }

    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time to free the MAP:  %19.10f", tcume);

      tstart = second();

      Cache_Aligned_Free((char *)dataa);
      Cache_Aligned_Free((char *)datab);

    tend = second();
    tcume = tend - tstart;
    ttotal = ttotal + tcume;
    printf ("\n Time to free the data arrays:  %19.10f", tcume);
    printf ("\n Total Time: %19.10f\n\n", ttotal);
}
```

U. EXAMPLE MAKEFILE (TAKEN FROM 4 BINS)

```
# ----------------------------------
# User defines FILES, MAPFILES, and BIN here
# ----------------------------------
FILES             = main.c

MAPFILES    = FourBinS.mc

BIN         = FourBinTest

# -----------------------------------
# User defined macros info supplied here
#
# (Leave commented out if not used)
# ------------------------------------
MACROS      = my_macro/fourbin.vhd
MY_BLKBOX   = my_macro/fourbin.box
MY_NGO_DIR  = my_macro
MY_INFO     = my_macro/fourbin.info
# ------------------------------------
# User supplied MCC and MFTN flags
# ------------------------------------

MY_MCCFLAGS       =
MY_MFTNFLAGS      =

# ------------------------------------
# User supplied flags for C & Fortran compilers
# ------------------------------------

CC          = icc       # icc for Intel cc for Gnu
FC          = ifc # ifc for Intel f77 for Gnu
LD          = ifc # ifc for Intel cc  for Gnu

MY_CFLAGS   =
MY_FFLAGS   =
# ------------------------------------
# No modifications are required below
# ------------------------------------
MAKIN   ?= $(MC_ROOT)/opt/srcci/comp/lib/AppRules.make

include $(MAKIN)
```

V. EXAMPLE PHASE SAMPLE INPUT FILE (32 INPUTS)

```
00

01

02

03

04

05

06

07

08

09

0A

0B

0C

0D

0E

0F

10
```

```
11
12
13
14
15
16
17
18
19
1A
1B
1C
1D
1E
1F
```

W. EXAMPLE RANGE BIN GAIN INPUT FILE (4 BINS)

```
2
1
2
1
```

X. EXAMPLE RANGE BIN PHASE ROTATION INPUT FILE (4 BINS)

```
1F
11
1F
11
```

Y. EXAMPLE SCREEN OUTPUT (4 BINS WITH 32 INPUTS)

```
Number of input samples: 32
 Time for disk access of input data:         0.0002677690
 Time to allocate the data caches for the MAP:         0.0000318932
 Time to pack the data for transfer to MAP:         0.0000015922
 Time for MAP allocation:         0.5351942658
 Time for MAP call:         0.0960569127
 Time to unpack results and send to output file:         0.0005680144
 Time to free the MAP:         1.0062198973
 Time to free the data arrays:         0.0000037104
 Total Time:        1.6383440550
```

Z. EXAMPLE OUTPUT DATA FILE (4 BINS WITH 32 INPUTS)

```
Iout  Qout  ODVout PSVout DRFM
----- ----- ------ ------ -----
00000 00000   0      1    00
00000 00000   0      1    01
00000 00000   0      1    02
00000 00000   0      1    03
0000F 0FFFC   1      1    04
00007 0FFFE   1      1    05
00016 0FFFB   1      1    06
0000E 0FFFF   1      1    07
0000E 00001   1      1    08
0000D 00005   1      1    09
0000B 00007   1      1    0A
0000A 0000A   1      1    0B
00007 0000C   1      1    0C
00005 0000D   1      1    0D
00002 0000E   1      1    0E
0FFFE 0000E   1      1    0F
0FFFB 0000E   1      1    10
0FFF7 0000D   1      1    11
0FFF5 0000B   1      1    12
0FFF2 0000A   1      1    13
0FFF0 00007   1      1    14
0FFEF 00005   1      1    15
0FFEE 00002   1      1    16
0FFEE 0FFFE   1      1    17
0FFEE 0FFFB   1      1    18
0FFEF 0FFF7   1      1    19
0FFF1 0FFF5   1      1    1A
0FFF2 0FFF2   1      1    1B
0FFF6 0FFF0   1      1    1C
0FFF8 0FFEF   1      1    1D
0FFFB 0FFEE   1      1    1E
0FFFF 0FFEE   1      1    1F
```

```
00001 0FFEE   1      0    00
00005 0FFEF   1      0    00
00007 0FFF1   1      0    00
0000A 0FFF2   1      0    00
0FFFD 0FFFA   1      0    00
00006 0FFFA   1      0    00
0FFF8 00000   1      0    00
00000 00000   0      0    00
00000 00000   0      0    00
00000 00000   0      0    00
```

LIST OF REFERENCES

[1] Kendrick R. Macklin, "Benchmarking and Analysis of the SRC-6E Reconfigurable Computing System," Master's Thesis, Naval Postgraduate School, Monterey, California, 2003.

[2] David Caliga and David Peter Barker, "Delivering Acceleration: The Potential for Increased HPC Application Performance Using Reconfigurable Logic," ACM 1-58113-293-X/01/0011, November 2001.

[3] Kai Kwang and Faye A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill, Inc., New York, 1984.

[4] "SRC-6E MAP© Hardware Guide," SRC-005-03, SRC Computers, Inc., Colorado Springs, January 6, 2003.

[5] "Virtex-II Platform FPGAs: Complete Data Sheet, DC and Switching Characteristics," DS031-3 (v3.1), Xilinx, Inc., San Jose, CA, October 14, 2003. From website: http://direct.xilinx.com/bvdocs/publications/ds031.pdf, accessed December 2003.

[6] "SRC-6E C Programming Environment V1.6 Guide," SRC-007-09, SRC Computers Inc., Colorado Springs, December 8, 2003.

[7] "SRC-6E Fortran Programming Environment V1.6 Guide," SRC-006-08, SRC Computers Inc., Colorado Springs, December 8, 2003.

[8] "SRC-6E MAP© Macro Developers Guide," SRC-008-01, SRC Computers Inc., Colorado Springs, September 23, 2002.

[9] "SRC-6E Programming Environment V1.6 Technical Note: Supported Macros," SRC Computers Inc., Colorado Springs, December 8, 2003.

[10] Douglas J. Fouts, Phillip E. Pace, Christopher Karow, and Stig R. T. Ekestorm, "A Single-Chip False Target Radar Image Generator for Countering Wideband Imaging Radars," *IEEE Journal of Solid-State Circuits*, Vol. 37, No. 6, pp. 751-759, 2002.

[11] Charles H. Roth, Jr., *Digital Systems Design Using VHDL*, PWS Publishing Company, Boston, 1998.

[12] Behrooz Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford University Press, New York, 2000.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Alan Hunsberger
   National Security Agency
   Ft. Meade, Maryland

4. Dr. Russell Duren
   Baylor University
   Engineering Department
   Rogers, Texas